

.NET Full Stack Development Program

LINQ & Thread

Outline

- LINQ
 - What is LINQ?
 - LINQ to Objects
 - Standard Query Operators
 - LINQ to SQL
- Thread
 - Thread & Process
 - Thread Class
 - Thread Life Cycle
 - Thread Problem / Thread Safety
 - Thread Synchronization
 - Dead Lock

Question

Top K Frequent Elements

Given an integer array `nums` and an integer `k`, return the `k` most frequent elements. You may return the answer in **any order**.

Example 1:

Input: `nums = [1,1,1,2,2,3]`, `k = 2`

Output: `[1,2]`

Example 2:

Input: `nums = [1]`, `k = 1`

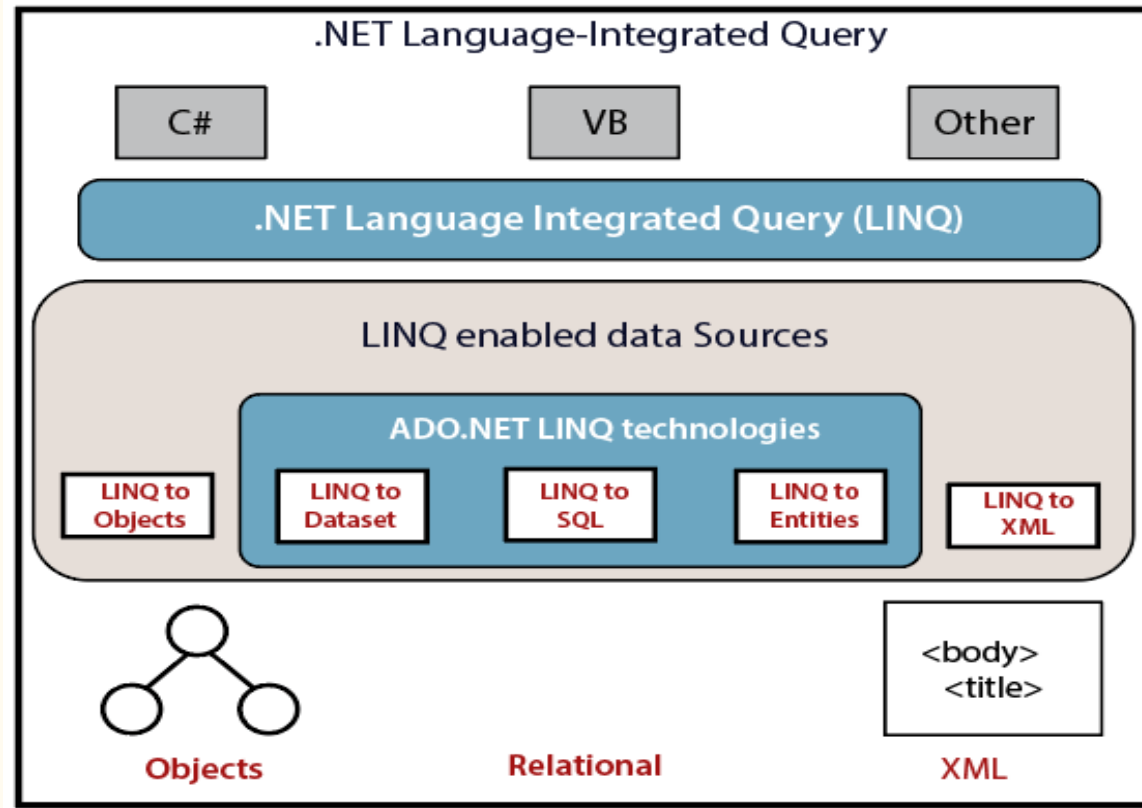
Output: `[1]`

LINQ

What is LINQ?

- LINQ(Language Integrated Query) provides us with common query syntax which allows us to query the data from various data sources.
- It was introduced by Microsoft with .NET Framework 3.5 and C# 3.0 and is available in **System.Linq** namespace.
- It supports a consistent query experience(LINQ Provider)
 - LINQ to objects
 - LINQ to SQL
 - LINQ to XML
 - ...

How does LINQ work?



LINQ Providers

- ❑ **LINQ to objects** (**No provider needed**): allows us to query in-memory objects from an array, collection and generics types.
- **LINQ to XML**(XLINQ): works with XML documents.
- **LINQ to SQL**(DLINQ): works with the SQL Server database.
- **LINQ to Datasets**: provides us the flexibility to query data cached in a Dataset.
- **LINQ to Entities**: is used to query any database(including SQL Server, Oracle, MySQL etc.)

LINQ to Objects

The term "LINQ to Objects" refers to the use of LINQ queries with any `IEnumerable` or `IEnumerable<T>` collection directly, without the use of an intermediate LINQ provider or API such as `LINQ to SQL` or `LINQ to XML`. You can use LINQ to query any enumerable collections such as `List<T>`, `Array`, or `Dictionary<TKey,TValue>`. The collection may be user-defined.

Advantages:

- They are more concise and readable, especially when filtering multiple conditions.
- They provide powerful filtering, ordering, and grouping capabilities with a minimum of application code.
- They can be ported to other data sources with little or no modification.

IEnumerable

- IEnumerable is **an interface that defines one method, GetEnumerator() which returns an IEnumerator type.**
- The IEnumerable interface is a type of iteration design pattern. It means we can **iterate on** the collection of the type IEnumerable.
- Basically, you can write LINQ queries for any type that supports **IEnumerable** or the generic **IEnumerable<T>** interface.

Different ways to write a LINQ

- Query Syntax
- Method Syntax
- Mixed Syntax (Query + Method)

```
form data in datasource  
where condition  
select data;
```

```
DataSource.ConditionMethod().OtherMethod();
```

```
(form data in datasource  
where condition  
select data).Method();
```

Query Syntax vs. Method Syntax

Most queries in the introductory Language Integrated Query (LINQ) documentation are written by using the LINQ declarative query syntax. However, the query syntax must be translated into method calls for the .NET common language runtime (CLR) when the code is compiled. These method calls invoke the standard query operators, which have names such as **Where**, **Select**, **GroupBy**, **Join**, **Max**, and **Average**. You can call them directly by using **method syntax** instead of query syntax.

There will be some queries that must be expressed as method calls. you must use a method call to express a query that retrieves the number of elements that match a specified condition. You also must use a method call for a query that retrieves the element that has the maximum value in a source sequence.

```
//Query syntax:
IEnumerable<int> numQuery1 =
    from num in numbers
    where num % 2 == 0
    orderby num
    select num;

//Method syntax:
IEnumerable<int> numQuery2 = numbers.Where(num => num % 2 == 0).OrderBy(n => n);
```

LINQ Queries

- All LINQ query operations consist of three distinct actions:
 1. Obtain the data source
 2. Create the query
 3. Execute the query

```
class IntroToLINQ
{
    static void Main()
    {
        // The Three Parts of a LINQ Query:
        // 1. Data source.
        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

        // 2. Query creation.
        // numQuery is an IEnumerable<int>
        var numQuery =
            from num in numbers
            where (num % 2) == 0
            select num;

        // 3. Query execution.
        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
    }
}
```

Create the Query

The query expression contains **three clauses**: from, where and select

The **from** clause specifies the data source, the **where** clause applies the filter, and the **select** clause specifies the type of the returned elements.

```
var queryLondonCustomers = from cust in customers
                             where cust.City == "London"
                             select cust;
```

Execute the Query

The actual execution of the query is **deferred** until you **iterate over the query variable** in a foreach statement.

```
// Query execution.  
foreach (int num in numQuery)  
{  
    Console.WriteLine("{0,1} ", num);  
}
```

Manners Of Execution

```
var evenNumQuery =  
    from num in numbers  
    where (num % 2) == 0  
    select num;  
  
int evenNumCount = evenNumQuery.Count();
```

```
List<int> numQuery2 =  
    (from num in numbers  
     where (num % 2) == 0  
     select num).ToList();  
  
// or like this:  
// numQuery3 is still an int[]  
  
var numQuery3 =  
    (from num in numbers  
     where (num % 2) == 0  
     select num).ToArray();
```

LINQ operators are divided into 2 categories:

- Deferred Execution

Deferred execution means that the operation is not performed at the point in the code where the query is declared. The operation is performed only when the query variable is enumerated, for example by using a foreach statement.

Almost all the standard query operators whose return type is `IEnumerable<T>`

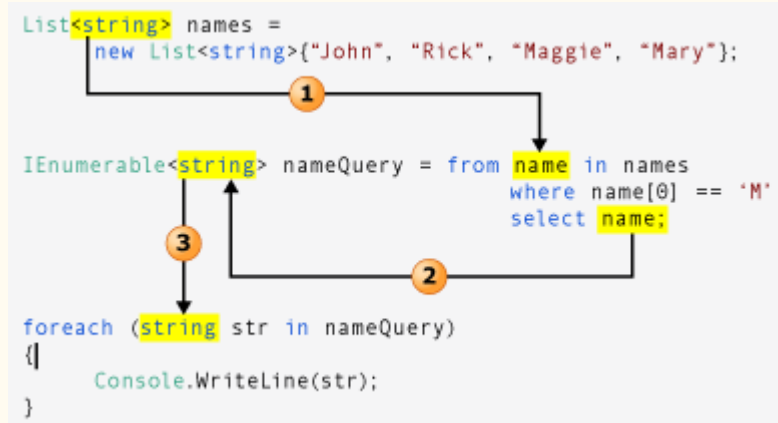
Ex: `Select()`, `SelectMany()`, `Where()`, etc.

- Immediate Execution

Immediate execution means that the data source is read and the operation is performed at the point in the code where the query is declared.

Ex: `ToArray()`, `ToList()`, `Aggregate Methods`: etc.

Type Relationships in LINQ



Standard LINQ Operators

Standard Query Operators

- The ***standard query operators*** are the **methods** that form the LINQ pattern. Most of these methods operate on sequences, where a sequence is an object whose type implements the **IEnumerable<T>** interface. The standard query operators provide query capabilities including **filtering, projection, aggregation, sorting and more.**
- Query Expression Syntax for Standard Query Operators:
 - Cast, GroupBy, GroupJoin, Join, OrderBy, OrderByDescending, Select, SelectMany, ThenBy, ThenByDescending, Where, and more

Classification	Standard Query Operators
Filtering	Where, OfType
Sorting	OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
Grouping	GroupBy, ToLookup
Join	GroupJoin, Join
Projection	Select, SelectMany
Aggregation	Aggregate, Average, Count, LongCount, Max, Min, Sum
Quantifiers	All, Any, Contains
Elements	ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
Set	Distinct, Except, Intersect, Union
Partitioning	Skip, SkipWhile, Take, TakeWhile
Concatenation	Concat
Equality	SequenceEqual
Generation	DefaultEmpty, Empty, Range, Repeat
Conversion	AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList

Filtering Data

Filtering refers to the operation of restricting the result set to contain only those elements that satisfy a specified condition

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query = from word in words
                           where word.Length == 3
                           select word;
```

```
int[] scores = { 60, 88, 90, 100, 75, 82, 96 };
IEnumerable<int> scoreQuerySyntax = from score in scores
                                   where score > 80
                                   select score;

IEnumerable<int> scoreMethodSyntax = scores.Where(score => score > 80);
```

Sorting Data

Sorting operations order the elements of a sequence based on one or more attributes:

OrderBy() | OrderByDescending() | ThenBy() | ThenByDescending()

```
string[] nameList = { "Emily", "Alice", "Alan", "Adam", "Bill", "Cindy", "Dave", "Eddie", "Zac" };  
//Sort by length first(des), if the length is the same sort by first letter(asc),  
//if the first letter is the same sort by second letter(des)  
IEnumerable<string> query = from name in nameList  
                           orderby name.Length descending, name.Substring(0,1), name.Substring(1,2) descending  
                           select name;
```

```
IEnumerable<string> method = nameList.OrderBy(name => name.Substring(0, 1))  
                                     .ThenBy(name => name.Length)  
                                     .ThenBy(name => name.Substring(1, 2));
```

Set Operators

- Distinct

```
IEnumerable<string> query = from planet in planets.Distinct()  
                           select planet;
```

Removes duplicate values from a collection.

- Except

```
IEnumerable<string> query = from planet in planets1.Except(planets2)  
                           select planet;
```

Returns the set difference, which means the elements of one collection that do not appear in a second collection.

- Intersect

```
IEnumerable<string> query = from planet in planets1.Intersect(planets2)  
                           select planet;
```

Returns the set intersection, which means elements that appear in each of two collections

- Union

```
IEnumerable<string> query = from planet in planets1.Union(planets2)  
                           select planet;
```

Returns the set union, which means unique elements that appear in either of two collections

Quantifier Operation

Quantifier operations return a **Boolean** value that indicates whether some or all of the elements in a sequence satisfy a condition

- All

Determines whether all the elements in a sequence satisfy a condition.

```
// Determine which market have all fruit names length equal to 5
IEnumerable<string> names = from market in markets
                           where market.Items.All(item => item.Length == 5)
                           select market.Name;
```

- Any

Determines whether any elements in a sequence satisfy a condition

```
// Determine which market have any fruit names start with 'o'
IEnumerable<string> names = from market in markets
                           where market.Items.Any(item => item.StartsWith("o"))
                           select market.Name;
```

- Contains

Determines whether a sequence contains a specified element

```
// Determine which market contains fruit names equal 'kiwi'
IEnumerable<string> names = from market in markets
                           where market.Items.Contains("kiwi")
                           select market.Name;
```


Projection Operation

Projection refers to the operation of transforming an object into a new form that often consists only of those properties that will be subsequently used.

- Select

Projects values that are based on a transform function

- SelectMany

Projects sequences of values that are based on a transform function and then flattens them into one sequence



Partitioning Data

Partitioning in LINQ refers to the operation of dividing an input sequence into two sections, without rearranging the elements, and then returning one of the sections

- Skip: Skips elements up to a specified position in a sequence
- SkipWhile: Skips elements based on a predicate function until an element does not satisfy the condition
- Take: Takes elements up to a specified position in a sequence
- TakeWhile: Takes elements based on a predicate function until an element does not satisfy the condition
- Chunk: Splits the elements of a sequence into chunks of a specified maximum size

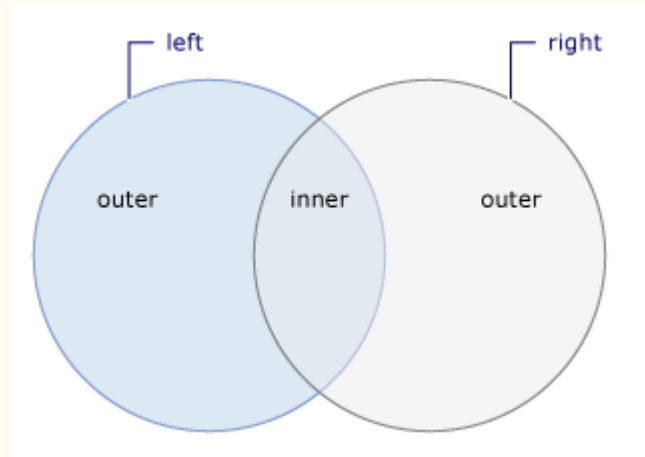
```
int chunkNumber = 1;
foreach (int[] chunk in Enumerable.Range(0, 8).Chunk(3))
{
    Console.WriteLine($"Chunk {chunkNumber++}:");
    foreach (int item in chunk)
    {
        Console.WriteLine($"    {item}");
    }

    Console.WriteLine();
}
```

```
// This code produces the following output:
// Chunk 1:
//    0
//    1
//    2
//
//Chunk 2:
//    3
//    4
//    5
//
//Chunk 3:
//    6
//    7
```

Join Operation

A **join** of two data sources is the association of objects in one data source with objects that share a common attribute in another data source.



Join Operator

Employee Table

Id	Name	Age	DeptNo
1	Judy	22	1
2	Alen	23	2

Department Table

Id	Name
1	Human Resources
2	Development

Employee Department Information Table

Id	Name	DeptName
1	Judy	Human Resources
2	Alen	Development

Join Operation

```
List<Employee> employeeList = new List<Employee>
{
    new Employee{ Id = 1, Name = "Judy", DeptNo = 1, Age = 22},
    new Employee{ Id = 2, Name = "Alden", DeptNo = 2, Age = 23},
    new Employee{ Id = 3, Name = "Lily", DeptNo = 1, Age = 22},
    new Employee{ Id = 4, Name = "Chris", DeptNo = 2, Age = 19},
    new Employee{ Id = 5, Name = "Ben", DeptNo = 1, Age = 20},
    new Employee{ Id = 6, Name = "Tom", DeptNo = 2, Age = 26},
    new Employee{ Id = 7, Name = "Dylan", DeptNo = 2, Age = 29},
    new Employee{ Id = 8, Name = "Zoey", DeptNo = 1, Age = 23},
};
```

```
List<Department> departmentList = new List<Department>
{
    new Department {Id = 1, Name = "Human Resources"},
    new Department {Id = 2, Name = "Development"},
};
```

```
//Join Query
var query = from e in employeeList
            join d in departmentList on e.DeptNo equals d.Id
            select new EmplDeptInfo
            {
                EmplId = e.Id,
                DeptName = d.Name,
                EmplName = e.Name,
            };
```

Join Operation

```
var method = employeeList.Join(departmentList,  
    e => e.DeptNo, // specify the first selector  
    d => d.Id,      // specify the second selector  
    (e, d) => new EmplDeptInfo  
    {  
        EmplId = e.Id,  
        DeptName = d.Name,  
        EmplName = e.Name,  
    });
```

Grouping Data

- GroupBy()/group by: takes a flat sequence of elements and then organizes the elements into groups, based on a given key, it will return an **IEnumerable<IGrouping<TKey, TSource>>** where **TKey** is nothing but the **Key** value on which the grouping has been formed and **TSource** is the collection of elements that matches the grouping key value.

```
List<int> numbers = new List<int>() { 35, 44, 200, 84, 3987, 4, 199, 329, 446, 208 };

IEnumerable<IGrouping<int, int>> query = from number in numbers
                                         group number by number % 2;

foreach (var group in query)
{
    Console.WriteLine(group.Key == 0 ? "\nEven numbers:" : "\nOdd numbers:");
    foreach (int i in group)
        Console.WriteLine(i);
}
```

```
var groupByExample = numbers.GroupBy(x => x%2);
```

```
Odd numbers:
35
3987
199
329
```

```
Even numbers:
44
200
84
4
446
208
```

```
*/
```

Aggregate Methods

- **Sum()**: This method is used to calculate the total(sum) value of the collection.
- **Max()**: This method is used to find the largest value in the collection.
- **Min()**: This method is used to find the smallest value in the collection.
- **Average()**: This method is used to calculate the average value(*double*) of the numeric type of the collection.
- **Count()**: This method is used to count the number of elements present in the collection.


```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

```
int sum = arr.Sum();
```

```
int max = arr.Max();
```

```
int min = arr.Min();
```

```
double avg = arr.Average();
```

```
int count = arr.Count();
```

LINQ to SQL

LINQ to SQL

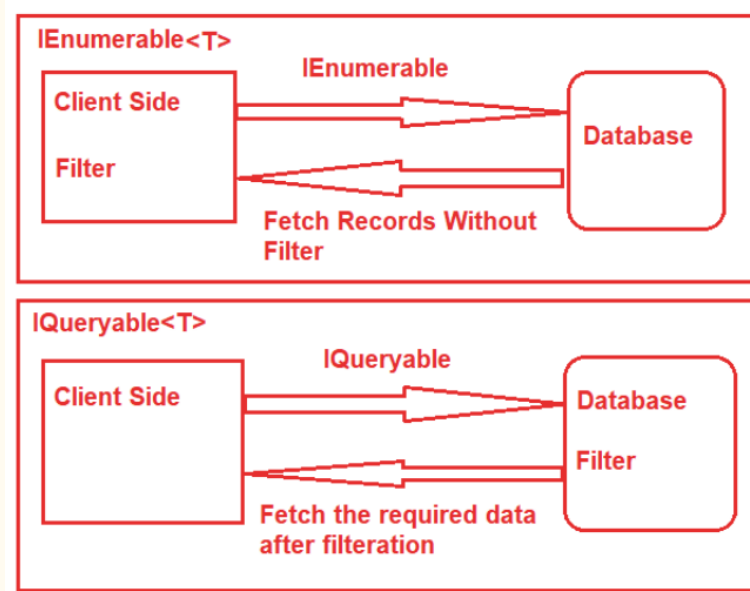
```
Table<Customer> Customers = db.GetTable<Customers>();  
  
IQueryable<string> custNameQuery =  
    from cust in Customers  
    where cust.City == "London"  
    select cust.Name;  
  
foreach (string str in custNameQuery)  
{  
    Console.WriteLine(str);  
}
```

The diagram illustrates the flow of data in the LINQ query. It shows three numbered steps: 1. The Customers table is retrieved. 2. The query is executed, filtering for customers in London. 3. The results are iterated over and written to the console.

IQueryable

- The IQueryable interface inherits the IEnumerable interface so that if it represents a query, the results of that query can be enumerated.
- We can use AsQueryable() and AsEnumerable() to convert them.

IEnumerable vs IQueryable



Question

Top K Frequent Elements

Given an integer array `nums` and an integer `k`, return the `k` most frequent elements. You may return the answer in **any order**.

Example 1:

Input: `nums = [1,1,1,2,2,3]`, `k = 2`

Output: `[1,2]`

Example 2:

Input: `nums = [1]`, `k = 1`

Output: `[1]`

Casting

```
class Plant
{
    public string Name { get; set; }
}

class CarnivorousPlant : Plant
{
    public string TrapType { get; set; }
}
```

```
static void Cast()
{
    Plant[] plants = new Plant[] {
        new CarnivorousPlant { Name = "Venus Fly Trap", TrapType = "Snap Trap" },
        new CarnivorousPlant { Name = "Pitcher Plant", TrapType = "Pitfall Trap" },
        new CarnivorousPlant { Name = "Sundew", TrapType = "Flypaper Trap" },
        new CarnivorousPlant { Name = "Waterwheel Plant", TrapType = "Snap Trap" }
    };

    var query = from CarnivorousPlant cPlant in plants
                where cPlant.TrapType == "Snap Trap"
                select cPlant;

    foreach (Plant plant in query)
        Console.WriteLine(plant.Name);

    /* This code produces the following output:

        Venus Fly Trap
        Waterwheel Plant
    */
}
```

Scenario: Find the set difference between two lists

```
class CompareLists
{
    static void Main()
    {
        // Create the IEnumerable data sources.
        string[] names1 = System.IO.File.ReadAllLines(@"../.././names1.txt");
        string[] names2 = System.IO.File.ReadAllLines(@"../.././names2.txt");

        // Create the query. Note that method syntax must be used here.
        IEnumerable<string> differenceQuery =
            names1.Except(names2);

        // Execute the query.
        Console.WriteLine("The following lines are in names1.txt but not names2.txt");
        foreach (string s in differenceQuery)
            Console.WriteLine(s);

        // Keep the console window open until the user presses a key.
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}

/* Output:
    The following lines are in names1.txt but not names2.txt
    Potra, Cristina
    Noriega, Fabricio
    Aw, Kam Foo
    Toyoshima, Tim
    Guy, Wey Yuan
    Garcia, Debra
    */
```


Scenario: Query for files with a specified attribute

```
class FindFileByExtension
{
    // This query will produce the full path for all .txt files
    // under the specified folder including subfolders.
    // It orders the list according to the file name.
    static void Main()
    {
        string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\";

        // Take a snapshot of the file system.
        System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

        // This method assumes that the application has discovery permissions
        // for all folders under the specified path.
        IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("*.*", System.IO.SearchOption.AllDirectories);

        //Create the query
        IEnumerable<System.IO.FileInfo> fileQuery =
            from file in fileList
            where file.Extension == ".txt"
            orderby file.Name
            select file;

        //Execute the query. This might write out a lot of files!
        foreach (System.IO.FileInfo fi in fileQuery)
        {
            Console.WriteLine(fi.FullName);
        }

        // Create and execute a new query by using the previous
        // query as a starting point. fileQuery is not
        // executed again until the call to Last()
        var newestFile =
            (from file in fileQuery
             orderby file.CreationTime
             select new { file.FullName, file.CreationTime })
            .Last();

        Console.WriteLine("\r\nThe newest .txt file is {0}. Creation time: {1}",
            newestFile.FullName, newestFile.CreationTime);
    }
}
```

Thread

- Thread
 - Thread & Process
 - Thread Class
 - Thread Life Cycle
 - Thread Problem / Thread Safety
 - Thread Synchronization
 - Dead Lock

Thread & Process

Process

- A process has a self-contained execution environment.

A process generally has a complete, private set of basic runtime resources; in particular, each process has its own memory space.

- Processes are often seen as synonymous with programs or applications
However, what the user sees as a single application may in fact be a set of cooperating processes.
 - Eg. To facilitate communication between processes, most operating systems support *Inter Process Communication* (IPC) resources, such as pipes and sockets

Process

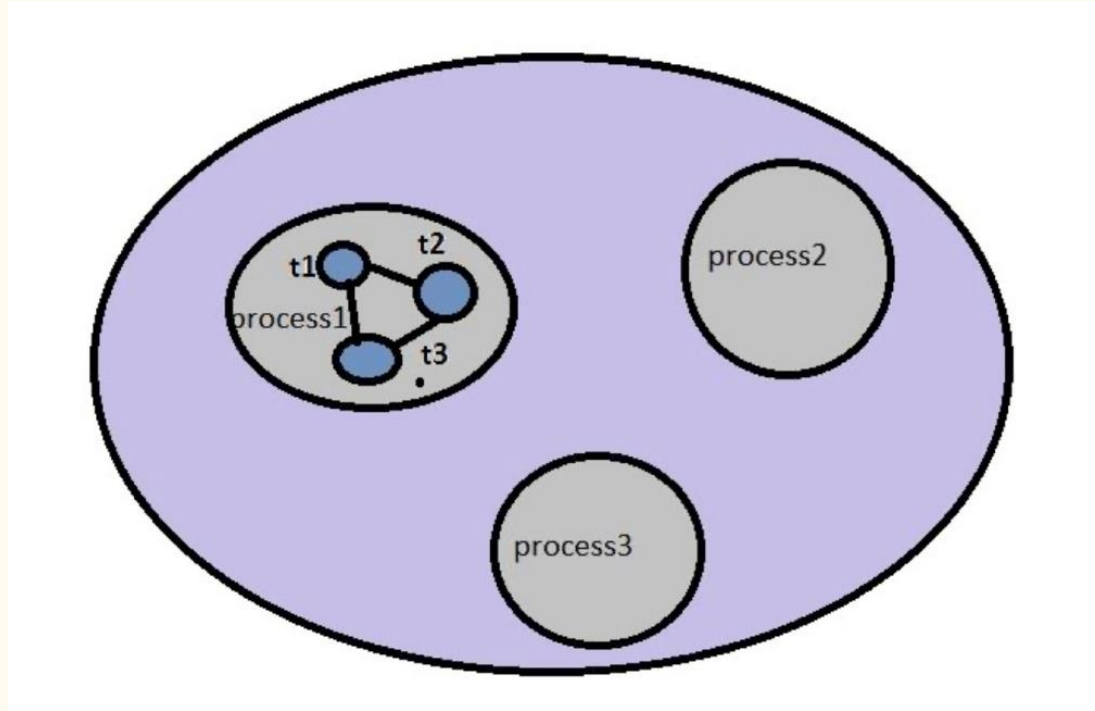
Task Manager

Processes

Run new task End task Efficiency mode View

Name	Status	9% CPU	77% Memory	1% Disk	0% Network
Apps (5)					
> Google Chrome (47)		0.8%	2,547.9 MB	0.1 MB/s	0.1 Mbps
> Microsoft PowerPoint		0.2%	122.7 MB	0 MB/s	0 Mbps
> Notion (11)		0.3%	687.0 MB	0.1 MB/s	0 Mbps
> Task Manager		0.9%	70.0 MB	0 MB/s	0 Mbps
> Windows Explorer		0%	158.9 MB	0 MB/s	0 Mbps
Background processes (107)					
> Adobe Acrobat Update Servic...		0%	0.1 MB	0 MB/s	0 Mbps
> Antimalware Service Executable		0.3%	113.8 MB	0 MB/s	0 Mbps
Application Frame Host		0%	0.8 MB	0 MB/s	0 Mbps
CefSharp.BrowserSubprocess		0%	0.5 MB	0 MB/s	0 Mbps
CefSharp.BrowserSubprocess		0%	0.3 MB	0 MB/s	0 Mbps
CefSharp.BrowserSubprocess		0%	0.6 MB	0 MB/s	0 Mbps
CefSharp.BrowserSubprocess		0%	0.3 MB	0 MB/s	0 Mbps
CefSharp.BrowserSubprocess		0%	2.2 MB	0 MB/s	0 Mbps

Thread vs. Process



Thread

- A thread is a lightweight sub-process, the smallest unit of processing. It has a separate path of execution.
- Threads are independent, if an exception occurs in one thread, it doesn't affect other threads.
- In other words, exceptions thrown in one thread cannot be handled by another thread.

Thread vs. Process

S.NO	PROCESS	THREAD
1.	Process means any program is in execution.	Thread means segment of a process.
2.	Process takes more time to terminate.	Thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	Process is less efficient in term of communication.	Thread is more efficient in term of communication.
6.	Process consume more resources.	Thread consume less resources.
7.	Process is isolated.	Threads share memory.

Why Multi-Threading?

- To enhance parallel processing
- To reduce response time to the user
 - Many servers use multithreads to achieve high performance
- To utilize the idle time of the CPU
 - Unit testing uses threads to run test cases in parallel
- Prioritize your work depending on priority
 - Computer games is a good example of multi-threading process (loading maps when you are working on other things)

Thread Class

Thread Class

- In C#, a multi-threading system is built upon the Thread class, which encapsulates the execution of threads.
- This class contains several methods and properties which helps in managing and creating threads and this class is defined under `System.Threading` namespace.
- The first thread to be executed in a process is called the **main** thread.
- When a C# program starts execution, the main thread is automatically created. The threads created using the **Thread** class are called the child threads of the main thread.
- Programmers can always take control of the main thread.

Thread Class

- The thread class provides lots of properties. Some of the important properties are as follows:
- **CurrentThread**: used to get the current running thread.
- **Name**: used to get or set the name of the thread.
- **Priority**: used to get or set the priority value(**Enum**) of the thread.
- **ThreadState**: used to get the thread state value(**Enum**) of the thread.
- **IsAlive**: returns a **bool** value representing whether or not this thread is alive
- **IsBackground**: used to get or set the value(**bool**) indicating whether the thread is a background thread or not.

Thread Class

- Create a Thread

```
static void Main(string[] args)
{
    //Step1: Create an instance of Thread
    Thread t1= new Thread(PrintHelloWorld);
    Thread t2 = new Thread(PrintNumber);

    //Step2: Run the Thread by calling the Start() method
    t1.Start();
    t2.Start(5);
}

1 reference
static void PrintHelloWorld()
{
    Console.WriteLine("Hello World");
}

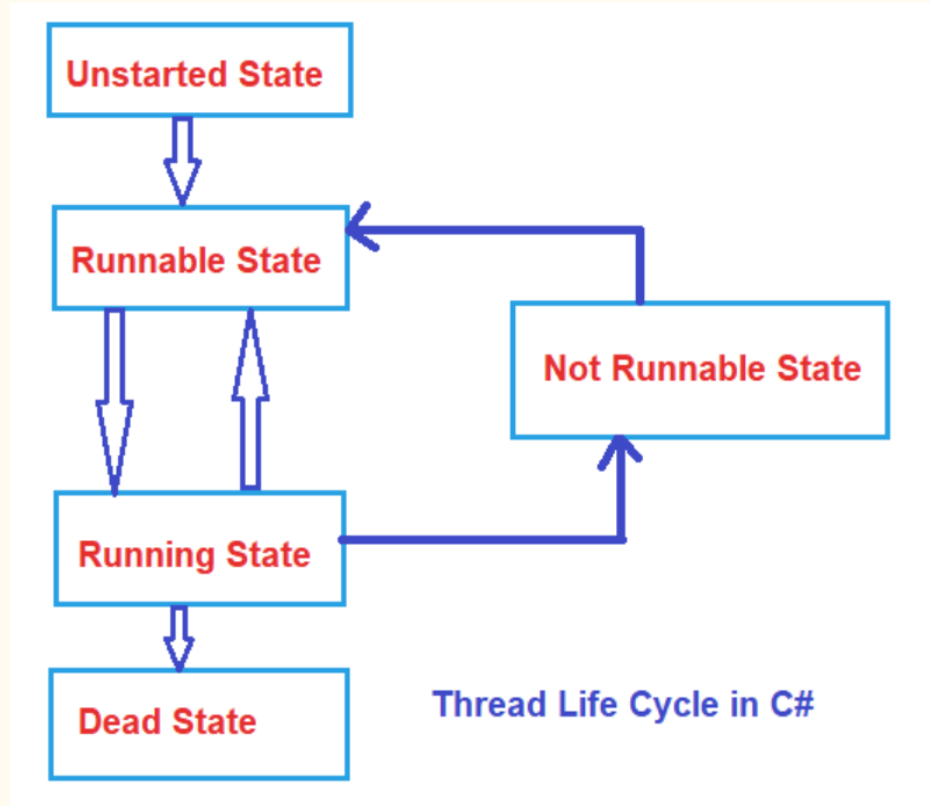
1 reference
static void PrintNumber(object num)
{
    Console.WriteLine(Convert.ToInt32(num));
}
```

Thread Life Cycle

The life cycle of a thread starts when an object of the Thread class is created and ends when the thread is terminated or completes execution.

- **The Unstarted State** – It is the situation when the instance of the thread is created but the Start method is not called.
- **The Runnable State** – It is the situation when the thread is ready to run(the Start method has been called).
- **The Running State** – The thread is running and not stopped yet.
- **The Not Runnable State** – A thread is not executable, when
 - Sleep method has been called
 - Join method has been called
 - Blocked by I/O operations
- **The Dead State** – It is the situation when the thread completes execution or is aborted by calling Abort() method.

Thread Life Cycle



Not Runnable State

- `Sleep()` – `Thread.Sleep()` causes the current thread to suspend execution for a specific milliseconds. `Sleep()` is a static method in `Thread` class.
- `Join()` – The `Join()` method blocks the calling thread until the thread represented by this instance terminates.
 - Example: We create the `Thread t` in the main thread;
 - Calling `t.Join()` will cause the main thread to wait for thread `t`.
- `Suspend()` – `Suspend()` method is called to suspend the thread. (deprecated)
- `Resume()` – `Resume()` method is called to resume the suspended thread.(deprecated)

Types of Thread

Types of Thread

- In C# we can create two types of threads in the application, they are:
 - Foreground Thread
 - Background Thread

Foreground Thread vs. Background Thread

- Foreground threads are those threads that keep running even after the main application exits or quits. So, the foreground threads do not care whether the main thread is alive or not, it completes only when it finishes its assigned work. That means the life of a foreground thread does not depend upon the main thread. Foreground thread is the **default type** when a new thread is created.
- Background Threads are those threads that will quit if our main thread is finished. The life of a background thread depends on the main thread.
 - A thread can be changed to a background thread at any time by setting it's **IsBackground** property to **true**.

Thread Safety

Thread Safety

- Threads communicate primarily by sharing access to the same resources such as fields or references to objects.
- This form of communication is extremely efficient, but it also makes some problems possible: *thread interference* and *data inconsistency*.

Thread Interference

- Consider a situation where two thread is operating on the same object at the same time.
- Interference happens when two operations, running in different threads, but acting on the same data, *interleave*. This means that the two operations consist of multiple steps, and the sequences of steps overlap.

```
class Counter
{
    2 references
    public int Value { get; set; } = 0;

    0 references
    public void Increment()
    {
        Value++;
    }

    0 references
    public void Decrement()
    {
        Value--;
    }
}
```

Thread Interference

The App will decompose the Increment method into following steps:

- Retrieve the current Value.
- Increment the retrieved value by 1.
- Store the incremented value back in the Property.

What if thread A is calling increment and thread B is calling decrement? (What will be the result?)

This situation is also called the Race Condition

```
class Counter
{
    2 references
    public int Value { get; set; } = 0;

    0 references
    public void Increment()
    {
        Value++;
    }

    0 references
    public void Decrement()
    {
        Value--;
    }
}
```

Data Inconsistency

- Data Inconsistency when different threads have inconsistent views of what should be the same data
- Eg. SameCounter class as before—Thread A increases the counter by 1, and thread B tries to print the value of counter at the same time. Now the value can either be 1 or 0

Thread Synchronization

- Synchronization can be achieved by using the **lock** keyword.
- It is used lock the object and only allow one thread to access the locked object, execute the task and then the lock will be released.
- It ensures that other thread does not interrupt the execution until the execution finish.
- The lock can only apply on objects.
- Syntax:

```
lock(object)
{
    //Statements to be synchronized
}
```

Thread-Safe Collections

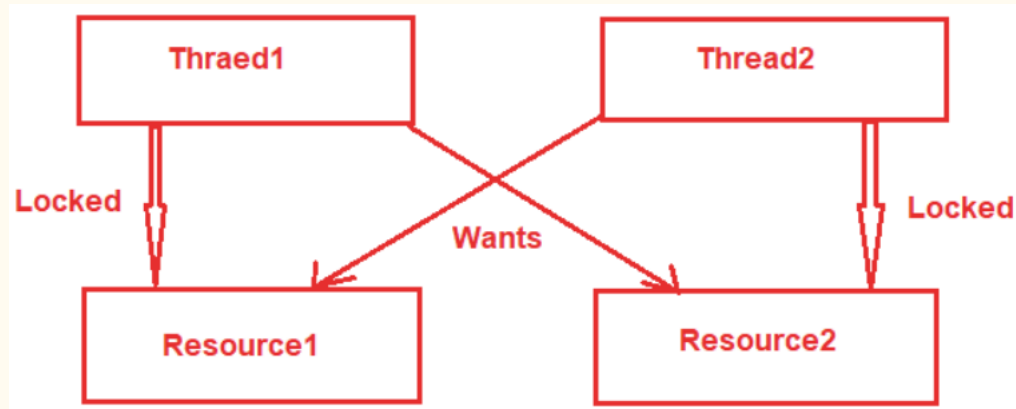
- The following table list some thread-safe collections under `System.Collections.Concurrent` namespace:

<code>ConcurrentDictionary<TKey,TValue></code>	Thread-safe implementation of a dictionary of key-value pairs.
<code>ConcurrentQueue<T></code>	Thread-safe implementation of a FIFO (first-in, first-out) queue.
<code>ConcurrentStack<T></code>	Thread-safe implementation of a LIFO (last-in, first-out) stack.

Deadlock

Deadlock

- *Deadlock* describes a situation where two or more threads are blocked forever, waiting for each other to release the lock.



Recap

- LINQ
 - What is LINQ?
 - LINQ to Objects
 - Standard Query Operators
 - LINQ to SQL
- Thread
 - Thread & Process
 - Thread Class
 - Thread Life Cycle
 - Thread Problem / Thread Safety
 - Thread Synchronization
 - Dead Lock

Any Questions?

.NET Full Stack Development Program

Day 8 SQL Server Intro

Outline

- Data Modeling
- ER Diagram
- Normalization
- Database
- SQL
- Constraints

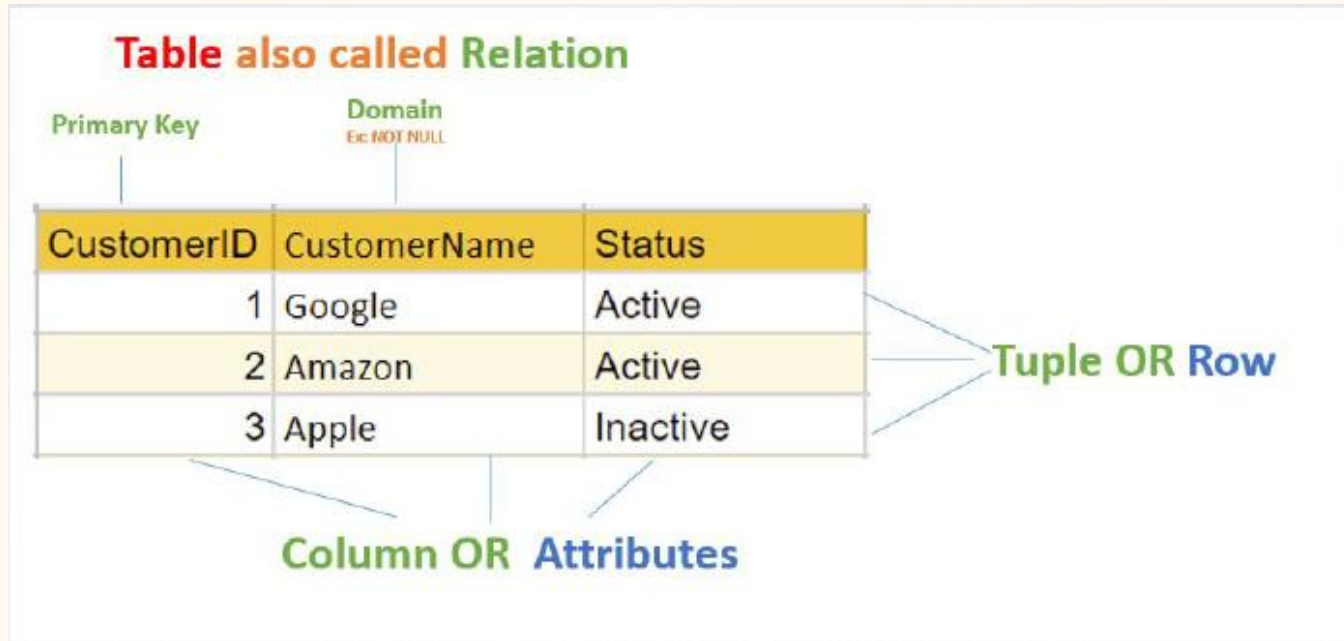
Data Modeling

- Data Model is the process of analyzing **business requirement**, designing and creating physical instance(a plan or a blueprint) for the database or data warehouse.
- It is a stage in SDLC(Software Development Life Cycle)
- It also impacts the user interface

Relational Database Terminology

SQL term	Relational database term	Description
Row	Tuple or record	A data set representing a single item
Column	Attribute or Field	A labeled element of a tuple, e.g. "Address" or "Date of birth"
Table	Relation or Base relvar	A set of tuples sharing the same attributes; a set of columns and rows
View or Result set	Derived relvar	Any set of tuples; a data report from the RDBMS in response to a query

Relational Database Terminology



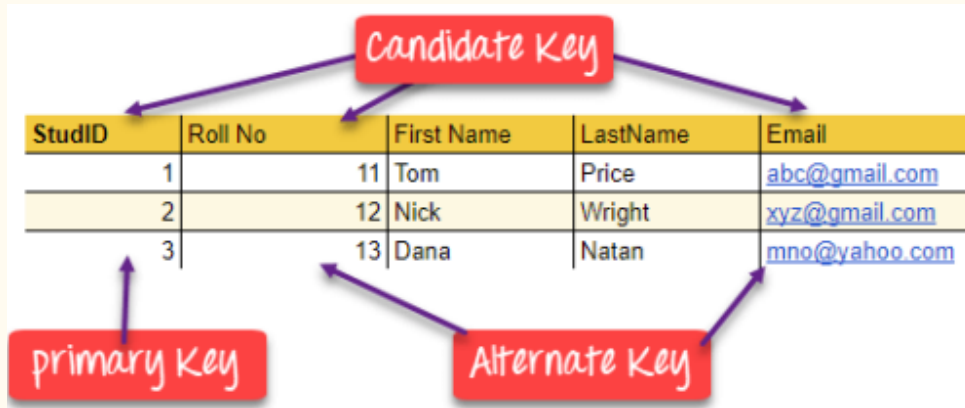
Data Modeling Terminology

- Entity
 - An item that can exist independently or uniquely identified
- Attribute
 - Column label(name)
- Domain
 - Set of valid values for an attribute
- Relationship
 - How entities relate
- Degree
 - How many entities in a relationship
- Cardinality
 - Measure of participation

Keys Used in Data Modeling

Super key	One or multiple attributes that can uniquely identify a row
Unique key / Candidate key	<ul style="list-style-type: none">• Unique identifier for a row• Minimal super key• No proper subset of candidate key can uniquely identify a row in the table
Primary key	<ul style="list-style-type: none">• Selected candidate key• One per table• Usually ID column(auto increment integer)
Alternate key	All candidate keys that are not primary key
Composite key	Superkey with two or more attributes
Foreign key	A foreign key is a column which is known as Primary key in the other table

Key Used in Data Modeling



StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

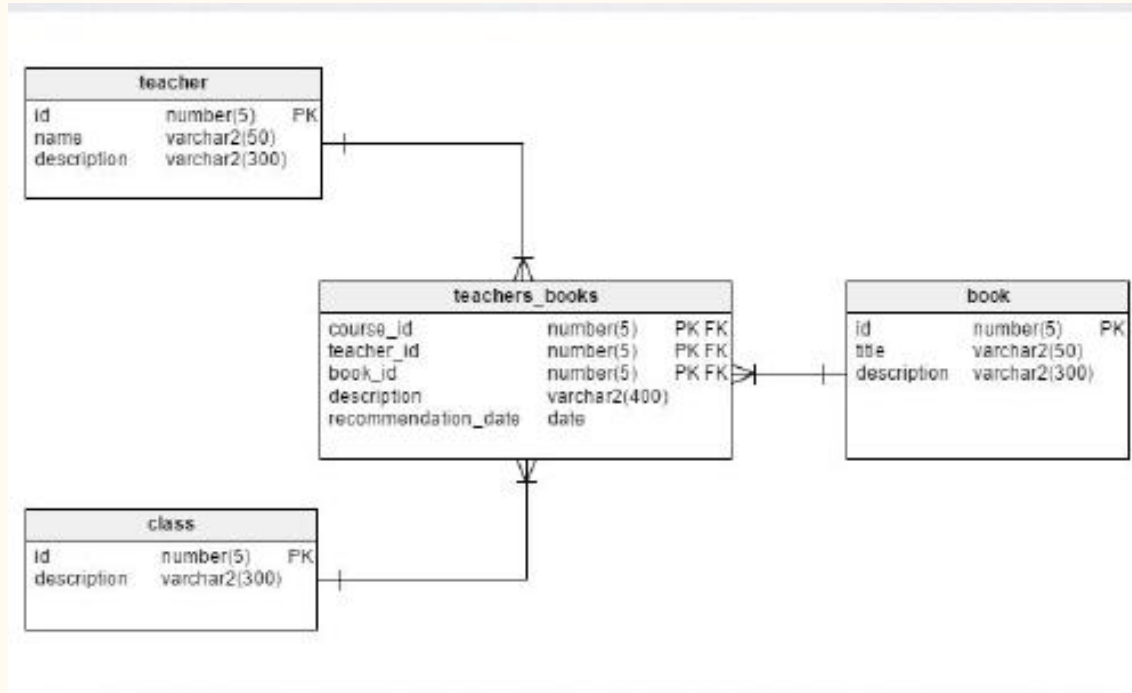
Suit	Value	No. times played
Hearts	Ace	5
Diamonds	Three	2
Hearts	Jack	3
Clubs	Three	5
Spades	Five	1

One is not enough to identify, but both combined make a unique value

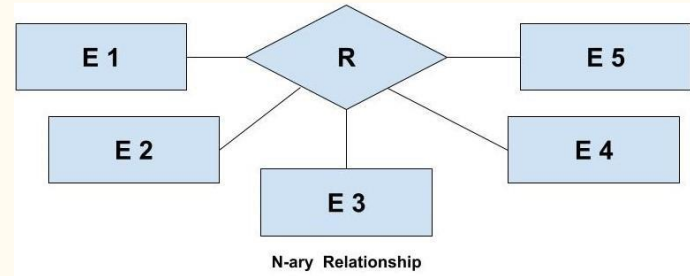
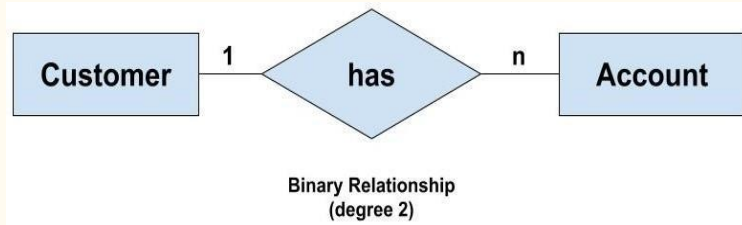
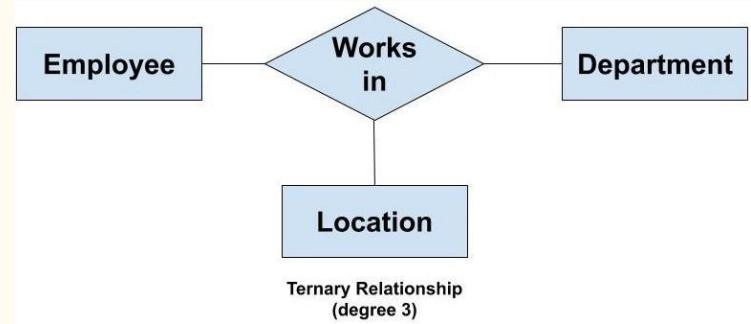
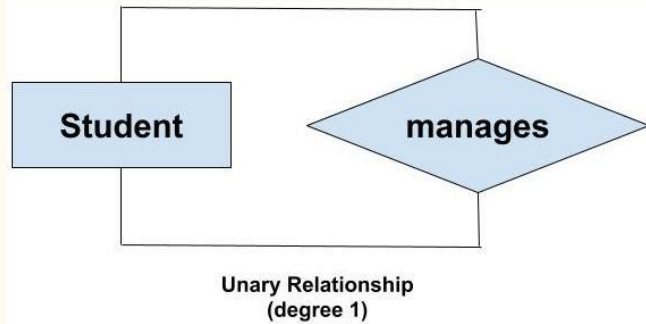
Cardinality and Degree

- Cardinality is the number of times the entity participates in the relationship
 - One-to-One: One element in entityA may link to one element in entityB and vice versa
 - One-to-Many: One element in entityA may link to many elements in entity but one element in entity may only link to one element in entityA
 - Many-to-One: Reverse A and B in one-to-many
 - Many-to-Many: One element in entityA may link to any number of elements in entity and vice versa
- Degree
 - Degree is the number of entities involved in the relationship and it is usually 2(binary relationship) however Unary and higher degree relationships can exists.
- Cardinality \neq Degree

Cardinality and Degree

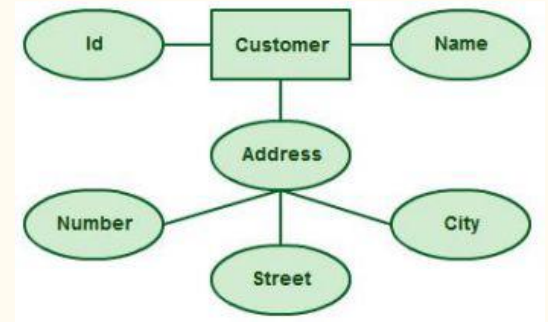


Cardinality and Degree



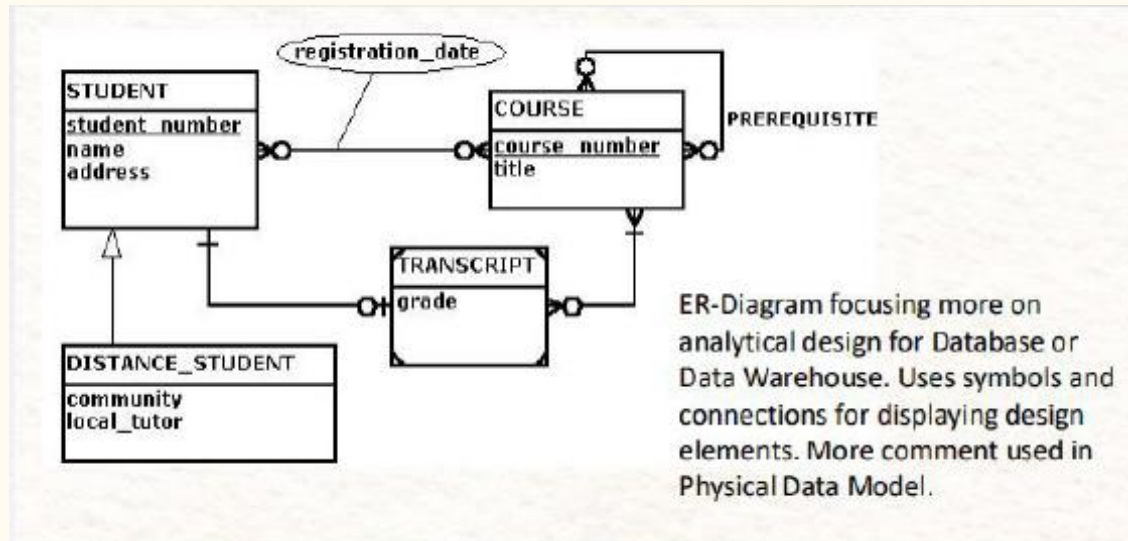
ERD

- ER-Diagram
 - Entity Relationship Diagram
 - Used to create or design a blueprint of the Database or Data Warehouse
 - Design Entities, attributes and show relationships







Crow's Foot Notation

- Connection Symbols display Relationships
- Entity and Attributes in Table like format



Crow's Foot Notation

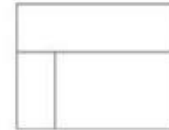
Symbol	Meaning
	One—Mandatory
	Many—Mandatory
	One—Optional
	Many—Optional



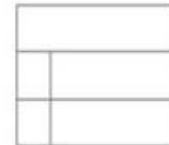
Entity
(with no attributes)



Entity
(with attributes field)

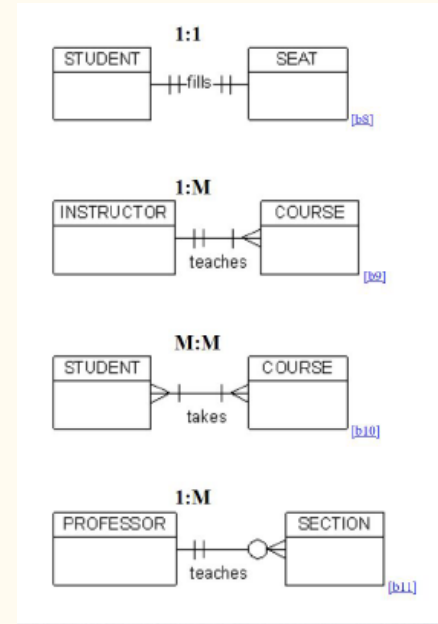
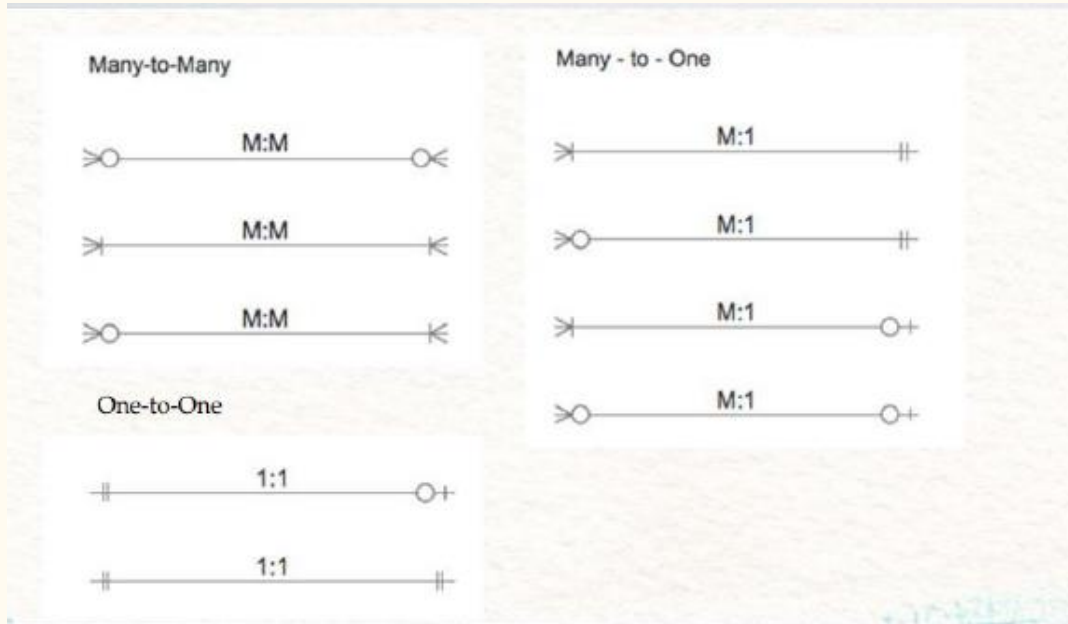


Entity
(attributes field with columns)



Entity
(attributes field with columns and
variable number of rows)

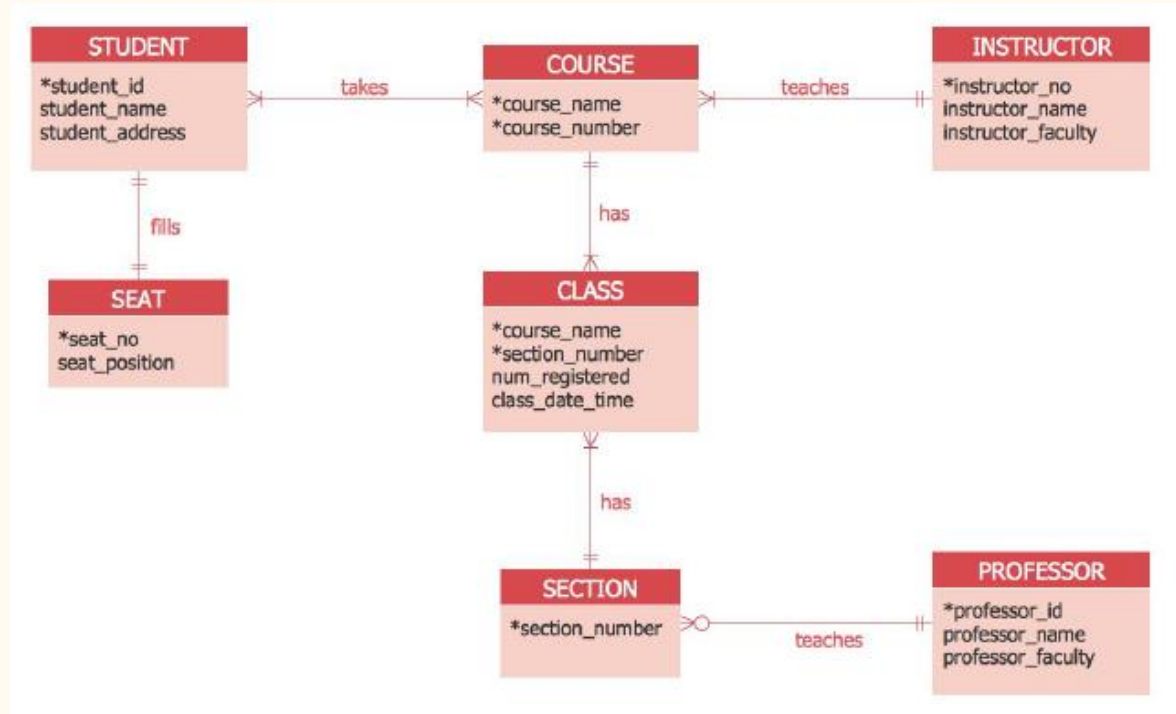
Crow's Foot Notation



Convert ERD to Tables

- Each entity type becomes a table
- Each single-valued attribute becomes a column
- Derived attributes are ignored/computed column
- Multi-valued attributes are represented by a separate table
- Use Conjunction Table to break up the Many-to-Many relationship
- The key attribute of the entity type becomes the primary key or unique key of the table

Convert ERD to Tables



Normalization

- Redundancy
 - Values repeated unnecessarily in multiple records or fields within one or more tables

Patient Id	Name	D.o.B	Gender	Phone	Doctor Id	Doctor	Room
134	Jeff	4-Jul-1993	Male	7876453	01	Dr Hyde	03
178	David	8-Feb-1987	Male	8635467	02	Dr Jekyll	06
198	Lisa	18-Dec-1979	Female	7498735	01	Dr Hyde	03
210	Frank	29-Apr-1983	Male	7943521	01	Dr Hyde	03
258	Rachel	8-Feb-1987	Female	8367242	02	Dr Jekyll	06

The table illustrates redundancy. The first row (Patient 134) and the third row (Patient 198) are both linked to a 'Duplicate' label, indicating they are redundant. Similarly, the fourth row (Patient 210) is also linked to a 'Duplicate' label, indicating it is redundant. The second and fifth rows (Patients 178 and 258) are not linked to any labels, indicating they are not redundant.

Normalization

- Anomaly
- When an attempt is made to modify(update, insert into, or delete from) a relation, the following undesirable side-effects may arise in relations that have not been sufficiently normalized
- Anomaly is the issue that may occur because of redundancy
- Types: Update, Insertion and Deletion

Normalization

- Anomaly Update
- The same information can be expressed on multiple rows; Therefore, updates to the relation may result in logical inconsistencies

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

Normalization

- Anomaly Insertion
- There are circumstances in which certain facts cannot be recorded at all

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

Handwritten red text: "No? NULL" with a red circle around the "Course Code" header.


424	Dr. Newsome	29-Mar-2007	?
-----	-------------	-------------	---

Normalization

- Anomaly Deletion
- Under certain circumstances, deletion of data representing certain facts necessitates deletion of data representing completely different facts

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201


DELETE

Normalization

- Main goal of normalization
- Reduce redundancy, avoid anomaly and create a well-structured series of table without error or inconsistencies
- Minimize redesign when extending the database structure
 - New type of data can be accommodated without changing existing structure too much
- Ensure data dependencies are properly enforced by data integrity constraints(Entity Integrity, Referential Integrity, Domain Integrity)

Normalization

- Functional Dependencies

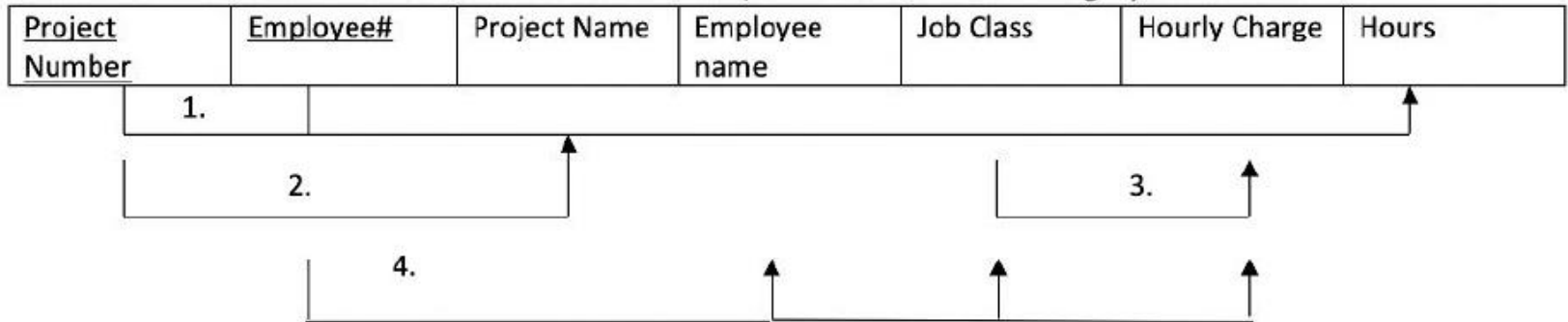
- Functional Dependencies are how different attributes relate in a table
- At this level, we focus on individual tables
- We see how individual attributes relate to the keys in the table
 - Primary Key & Candidate Keys = Prime Attributes
 - Attributes that aren't keys = Non-Prime Attributes

- Types of Dependencies

- Full Dependencies – Depends on all prime attributes fully
- Partial Dependencies – Depends on some Prime Attributes
- Transitive Dependencies – Depends on an attribute that depends on a Prime Attribute

Normalization

1. The diagram below shows a relational schema and its functional dependencies. Label each dependency as a full, partial or transitive dependency. Convert the relation to a set of relations in second normal form. Then show the relations in 3rd normal form to illustrate the progression from a single relation to 2nd normal form and then 3rd normal form. For the set of relations in 3rd normal form, show the referential integrity constraints.

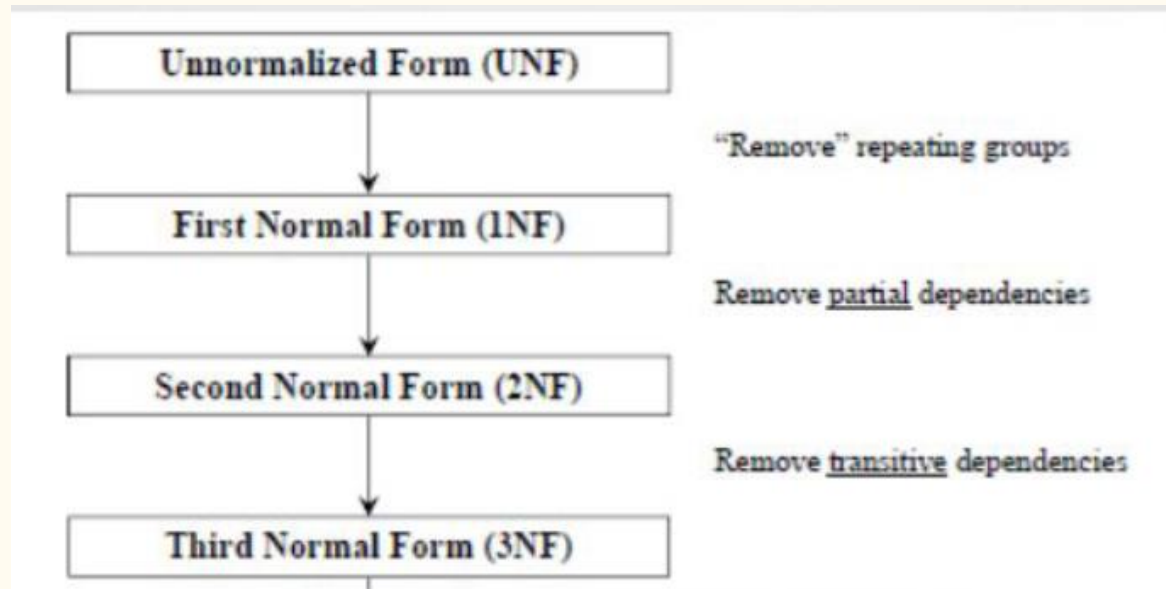


Normalization

- First Normal Form
 - Each table cell should contain a single value
 - Each record needs to be unique
- Second Normal Form
 - Meets all of 1NF
 - Makes sure all non-prime attributes are fully dependent on a prime attribute
- Third Normal Form
 - Meets 1NF and 2NF
 - Every non-prime attribute is non-transitively dependent on the prime attributes

Normalization

- Process



Normalization

- UNF

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	\$84.50	23.80
		101	John G. News	Database Designer	\$105.00	19.40
		105	Alice K. Johnson	Database Designer	\$105.00	35.70
		106	William Smithfield	Programmer	\$35.75	12.60
		102	David H. Senior	Systems Analyst	\$96.75	23.80
18	Amber Wave	114	Annelise Jones	Applications Designer	\$48.10	24.60
		118	James J. Frommer	General Support	\$18.36	45.30
		104	Anne K. Ramoras	Systems Analyst	\$96.75	32.40
		112	Darlene M. Smithson	DSS Analyst	\$45.95	44.00
22	Rolling Tide	105	Alice K. Johnson	Database Designer	\$105.00	64.70
		104	Anne K. Ramoras	Systems Analyst	\$96.75	48.40
		113	Delbert K. Joenbrood	Applications Designer	\$48.10	23.60
		111	Geoff B. Wabash	Clerical Support	\$26.87	22.00
		106	William Smithfield	Programmer	\$35.75	12.80
25	Starflight	107	Maria D. Alonzo	Programmer	\$35.75	24.60
		115	Travis B. Bawangi	Systems Analyst	\$96.75	45.80
		101	John G. News	Database Designer	\$105.00	56.30
		114	Annelise Jones	Applications Designer	\$48.10	33.10
		108	Ralph B. Washington	Systems Analyst	\$96.75	23.60
		118	James J. Frommer	General Support	\$18.36	30.50
		112	Darlene M. Smithson	DSS Analyst	\$45.95	41.40

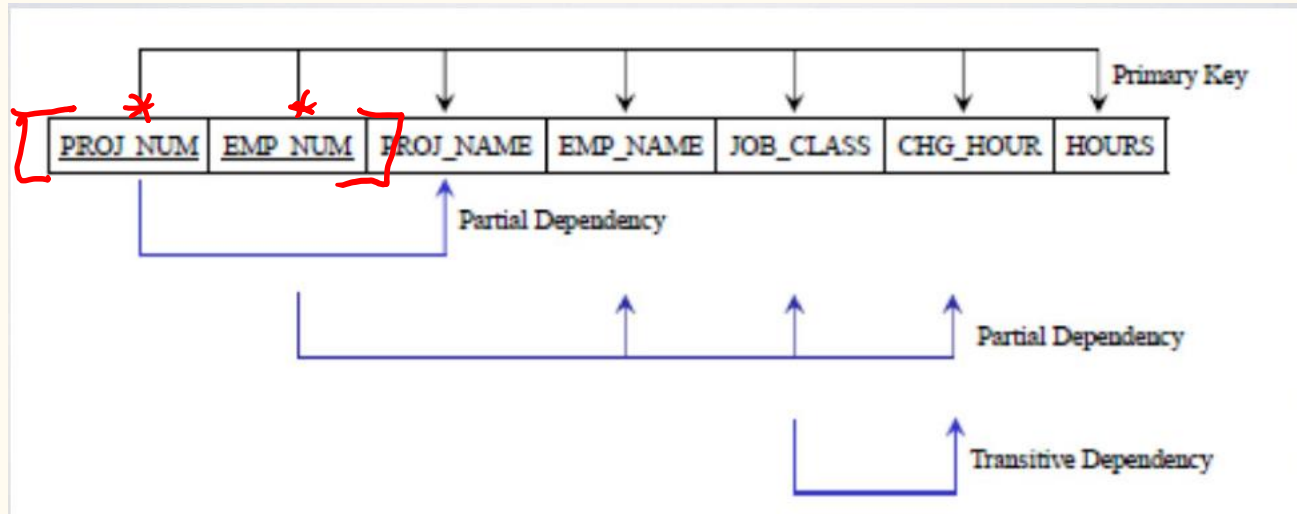
Normalization

- After 1NF

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	\$84.50	23.80
15	Evergreen	101	John G. News	Database Designer	\$105.00	19.40
15	Evergreen	105	Alice K. Johnson	Database Designer	\$105.00	35.70
15	Evergreen	106	William Smithfield	Programmer	\$35.75	12.60
15	Evergreen	102	David H. Senior	Systems Analyst	\$96.75	23.80
18	Amber Wave	114	Annelise Jones	Applications Designer	\$48.10	24.60
18	Amber Wave	118	James J. Frommer	General Support	\$18.36	45.30
18	Amber Wave	104	Anne K. Ramoras	Systems Analyst	\$96.75	32.40
18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	\$45.95	44.00
22	Rolling Tide	105	Alice K. Johnson	Database Designer	\$105.00	64.70
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	\$96.75	48.40
22	Rolling Tide	113	Delbert K. Joenbrood	Applications Designer	\$48.10	23.60
22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	\$26.87	22.00
22	Rolling Tide	106	William Smithfield	Programmer	\$35.75	12.80
25	Starflight	107	Maria D. Alonzo	Programmer	\$35.75	24.60
25	Starflight	115	Travis B. Bawangi	Systems Analyst	\$96.75	45.80
25	Starflight	101	John G. News	Database Designer	\$105.00	56.30
25	Starflight	114	Annelise Jones	Applications Designer	\$48.10	33.10
25	Starflight	108	Ralph B. Washington	Systems Analyst	\$96.75	23.60
25	Starflight	118	James J. Frommer	General Support	\$18.36	30.50
25	Starflight	112	Darlene M. Smithson	DSS Analyst	\$45.95	41.40

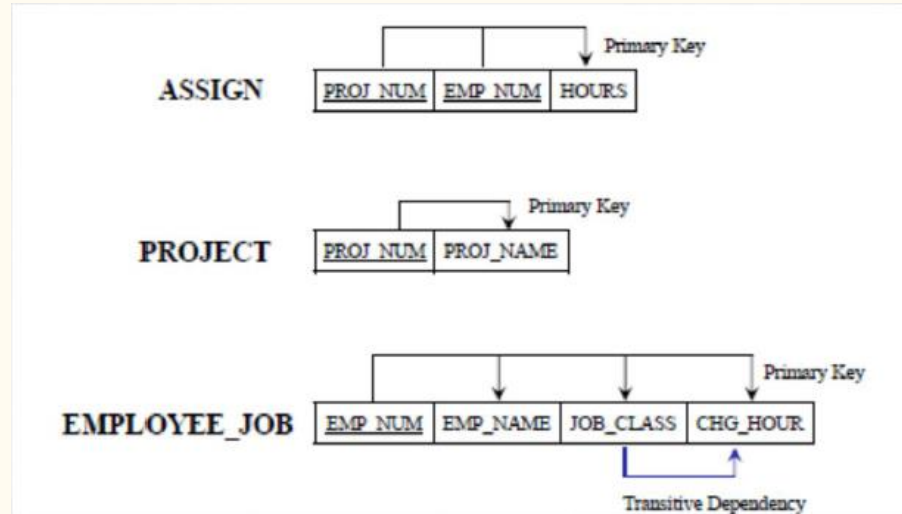
Normalization

- Dependencies



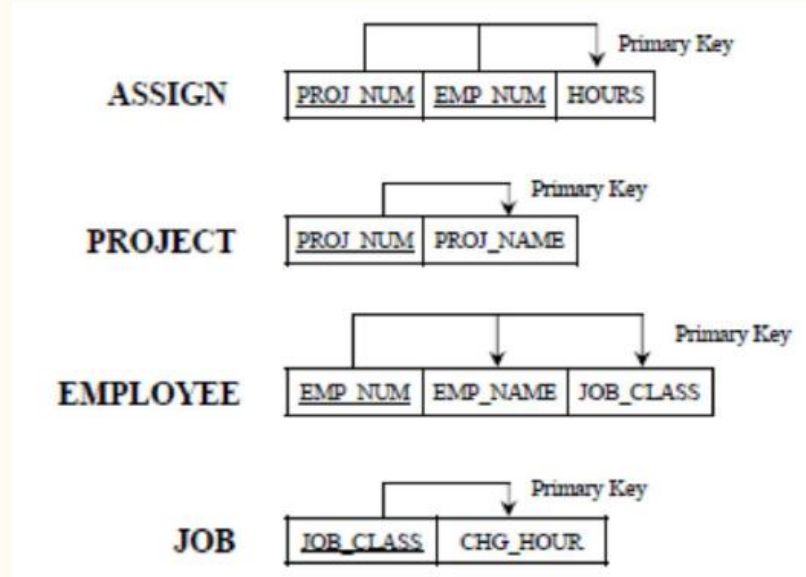
Normalization

- After 2NF(Remove partial dependencies)



Normalization

- After 3NF(Remove transitive dependency)



Normalization

- When do we normalize?
 - Usually normalize in database with lots of read/writes
 - Not too much fetching/only need to fetch a small subset of the data
 - Data integrity is crucial especially because there us a lot of user input
- Cons
 - Normalization is an expensive process
 - Designing can be difficult – Good for final design, not testing
 - More tables leads to more time
 - Joins are costly, having more tables can cause slowing

Database Integrity

- Entity Integrity
 - Design of the table or entity
 - String PK with no nulls or repeats
- User-Defined Integrity
 - Rules or constraints applied by the user to maintain rules of design
- Domain Integrity
 - Correct and proper domains specified with proper use of columns
- Referential Integrity
 - Proper FK setup with proper PK reference
 - Good design for connection and joins

Database Management System



Relational Database vs. Non-Relational Database

- Relational Database(SQL)
 - Traditional way of storing data
 - Data is stored in tables with rows and columns
 - Rigid schema with well-defined relationships
 - Difficult to scale
 - Examples: SQL Server, Oracle, MySQL
- Non-Relational Database(NoSQL)
 - Developed more recently
 - Data can be stored in a variety of formats(JSON documents, key-value pairs, wide-column, graphs)
 - Flexible schema with loose relationships
 - Easily scalable
 - Examples: MongoDB, Redis

SQL Introduction

SQL(Structured Query Language)

- Data Definition Language(DDL)
- (DML)
- (DCL)
- (DQL)

SQL

- DDL – Data Definition Language
- Create
- Alter
- Drop
- Truncate

```
--Create a table
CREATE TABLE Employee
(
    Employee_Id INT PRIMARY KEY,
    First_Name CHAR(20) NOT NULL,
    Last_Name CHAR(20) NOT NULL
);

--ADD a column to a table
ALTER TABLE Employee
ADD Middle_Name CHAR(20);

--TRUNCATE a table
TRUNCATE TABLE Employee;

--DROP a table
DROP TABLE Employee;
```

Create

- This command builds a new table and has a predefined syntax.

--Syntax:

```
CREATE TABLE [table_name] ([column_definitions])[table_params];
```

```
--Create a table
CREATE TABLE Employee
(
    Employee_Id INT PRIMARY KEY,
    First_Name CHAR(20) NOT NULL,
    Last_Name CHAR(20) NOT NULL
);
```

- In this example, the string CHAR is used to specify the data type. Other data types can be DATE, INT, DECIMAL etc.

Alter

- An Alter command modifies an existing database table. This command can add up additional column, drop existing columns and even change the data type of columns involved in a database table.

--Syntax:

```
ALTER [object_type] [object_name] parameters;
```

```
ALTER TABLE Products  
ALTER COLUMN Product_Id INT NOT NULL;  
  
ALTER TABLE Products  
ADD PRIMARY KEY(Product_Id);
```

- In this example, we added a unique primary key to the table to add a constraint and enforce a unique value. The constraint “Product_Id” is a primary key and is on the Products table.

Drop

- A drop command is used to delete objects such as a table, index or view. A DROP statement **cannot be rolled back**, so once an object is destroyed, there's no way to recover it.

```
--Syntax:
```

```
DROP [object_type] [object_name];
```

```
--DROP a table
```

```
DROP TABLE Employee;
```


Truncate

- Similar to DROP, the TRUNCATE statement is used to quickly remove all records from a table. However, unlike DROP that completely destroys a table, TRUNCATE preserves its full structure to be reused later.

```
--Syntax:  
TRUNCATE TABLE [table_name];
```

```
TRUNCATE TABLE Products;
```

SQL

- DML – Data Manipulation Language
- Insert
- Update
- Delete

DML

- Insert Statements

- Insert statements are used to input data into tables
- The order of data specified should match order of columns
- If you don't know the order of columns, specify each column name in the insert statement

- For example

- Inserting data matching order
 - `Insert into TableName values(1, 'Name'),(2, 'Name');`
- Inserting data without knowing matching order
 - `Insert into TableName(Col2, Col1) values('Name', 1),('Name', 2);`

DML

- Update Statements

- Update statements are used to change or modify data inside a table
- Specify single row depending on statement
- Use unique identifiers to get correct rows

- For Example

- Update using unique column
 - Update TableName Set Name = 'Tim' Where ID = 2;
- Update using non-unique column
 - Update TableName Set Name = 'Hal' Where Name = 'Bob'

DML

- Delete Statements

- Delete statements are used to remove specific rows inside a table
- Use unique values to identify the correct rows to delete
- Delete leave logs and continue identity values

- For Example

- Deleting rows using specific unique values
 - Delete from TableName Where ID = 1;
- Deleting rows using non-unique values
 - Delete From TableName Where Name = 'Jim'

SQL

- DCL – Data Control Language
- DCL is syntax used to control what permission a user can have
- You can create Roles that users can be grouped into a specific permission there
- You can choose what tables someone in a role can access and even what statements can be performed

DCL

- Grant
 - Used to “grant” or give permissions to a user or role
- Example:
 - Create table permission to a role
 - Grant Create Table to TestRole
 - Add user to the role
 - Exec sp_addrolemember ‘TestRole’, ‘TestUser’

DCL

- Revoke

- Revoke is used to take away permissions given, whether they are Grant or Deny permissions.

```
REVOKE [GRANT OPTION FOR] [permission]
ON [object]
FROM [user]
[CASCADE]
```

- Example

- Revoking grant permission
- Revoke Create Table to TestRole

```
REVOKE SELECT
ON HR.employees
FROM Joe
```


DCL

- Deny
- Deny is preventing someone from having access at all.
- Deny is not the same as revoke, as revoke is made to take back, deny is made to say NO
- If grant and deny permission are given to the same role, deny will take over
- Example
 - Deny Create Table to TestRole

```
DENY [permission]  
ON [object]  
TO [user]
```

```
DENY DELETE  
ON HR.employees  
TO Matthew
```

SQL

- DQL – Data Query Language
- Select From Where
 - Select: Pick up which column of data you'd like to fetch
 - From: Select which table or data set to fetch.
 - Where: Specific a criteria to sort data by use operators(Filter).
 - Operators: In, Or, And
- Group by, Having, Order by
 - Group by – Used to combine similar values in column
 - Having – filter conditions for aggregate only
 - Order by – display the data by order by a specific column

```
SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5  
ORDER BY COUNT(CustomerID) DESC;
```

DQL Execution Order

Logic	Actual
SELECT	FROM
FROM	WHERE
WHERE	GROUP BY
GROUP BY	HAVING
HAVING	SELECT
ORDER BY	ORDER BY

Constraint

- A constraint is usually associated with a table and is created with a `CREATE CONSTRAINT` SQL statement.
- They define certain properties that data in a database must comply with

Constraint

- Key Constraints

- Primary Key
 - 1 per table
 - Unique Clustered index
 - Not Null
- Unique Key
 - 999 per table
 - Unique Non-Clustered index
 - 1 Null Allowed
- Foreign Key
 - Cannot exist before PK
 - Must be deleted before PK

- Other Constraints

- Null, Not Null
 - Are nulls allowed
- Check
 - Data must meet rule
- Default
 - If nothing, then this
- Data types
 - Char(2) – States (NY, CA...)
 - Varchar(10) – Names...
 - Money -- Money

Constraint

- Adding constraints to a table
 - Constraints: used to specify rules for the data in table
 - Type: NOT NULL, UNIQUE, PRIMARY KE
- How?
 - Create table then alter to add constraints
 - Add constraint as we specify the column in a table
 - Add constraint in the same create statement, after we specified the table

Constraint

- Add constraint as we specify the column in a table

```
CREATE TABLE Employee (  
    Emp_Id INT PRIMARY KEY,  
    First_Name CHAR(20) NOT NULL,  
    Last_Name CHAR(20),  
    Age INT  
)
```

- Add constraint in the same create statement after we specified the table

```
ALTER TABLE Employee  
ADD CONSTRAINT Fk_Dpet_Id FOREIGN KEY(Department_Id)  
REFERENCES Department(Id);
```

- Create a table then alter it to add constraints

```
CREATE TABLE Person(  
    ID INT NOT NULL IDENTITY,  
    First_Name CHAR(20) NOT NULL,  
    Last_Name CHAR(20),  
    Age INT,  
    CONSTRAINT Pk_Person PRIMARY KEY(ID, Last_Name)  
)
```

Questions?

.NET Full Stack Development Program

Day9 T-SQL Query

Outline

- Basic SQL Query
- Aggregate Function
- Join
- Sub-query & Set Operator
- View
- Variable
- Procedure
- Index

Select From Where

- The **SELECT** statement is used to select data from a database.
- The **WHERE** clause is used to filter records
- The data returned is stored in a result table, called the result set.
- Example:

```
--Syntax:  
SELECT col1, col2, ... , colN  
FROM table_name  
WHERE condition(boolean_expression)
```

SELECT Clauses

Attribute	Example	Explanation
*	*	All attributes from all relations
<table name>.*	FREQUENTS.*	All the attributes from relation <table name>
<alias name>.*	f.*	All the attributes from the relation aliased to f
Attribute list	d.lastName, d.firstName	Only the specified attributes
<Math equation>	1 + 3	Evaluates the expression
<constant>	'CPA' 3	Returns the specified constant

AS

- The AS command is used to create a more meaningful name by renaming a column or table with alias.
- It's optional and only exists for the duration of the query.

```
SELECT Product_Id AS Id, Product_Name AS Product  
FROM Products;
```

More SELECT Clauses

- Functions can be used in SELECT statement

Attribute	Example	Explanation
<function>	GETDATE()	returns current datetime
<function>	CONCAT()	Concatenates two or more string values in an end to end manner and returns a single string.

- More build-in functions:
- <https://learn.microsoft.com/en-us/sql/t-sql/functions/functions?view=sql-server-ver15>

WHERE Clause

1. `<attribute> = [value]`
2. `<attribute> BETWEEN [value1] AND [value2]`
3. `<attribute> IN ([value1], [value2], ...)`
4. `<attribute> LIKE 'SST%'`
5. `<attribute> LIKE 'SST_'`
6. `<attribute> IS NULL` AND `[attribute] IS NOT NULL`
7. Logical combinations with AND and OR
8. Conditional Operators `>`, `<`, `=`, `!=`
9. Subqueries ...

WHERE Clause

- Conditional Operators

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<> or !=	Not equal. In some databases, the <code>!=</code> is used to compare values which are not equal.
BETWEEN	Between some range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Aggregate Functions

- Aggregate functions are build-in functions that used to perform simple statistics
 - COUNT()
 - AVG()
 - SUM()
 - MAX()
 - MIN()
- The COUNT() function returns the number of rows that matches a specified criterion.
- The AVG() function returns the average value of a numeric column.
- The SUM() function returns the total sum.
- The MIN() and MAX() functions return the smallest and largest values of the selected columns respectively.

Aggregate Functions

- How does aggregate functions handle Null value?
 - COUNT(*) gives the total number of records in the table including Null values, there is no difference between these two functions (*can be replaced by any number in the int capacity)
 - COUNT(column_name) only considers rows where the column contains a Not-Null value
 - AVG, MIN, MAX, etc. ignore Null values
 - GROUP BY includes a row for null.

<https://learn.microsoft.com/en-us/sql/t-sql/functions/count-transact-sql?view=sql-server-ver16>

Group By

- In SQL Server, the GROUP BY clause is used to form the groups of records
- Any non-aggregate columns called in the select statement must be in group by
- Optional

	DRINKER	COFFEE	SCORE
1	Risa	Espresso	2
2	Chris	Cold Brew	1
3	Chris	Turkish Coffee	5
4	Risa	Cold Brew	4
5	Risa	Cold Brew	5

```
SELECT r.COFFEE, AVG (r.SCORE)
FROM RATES r
GROUP BY r.COFFEE;
```

Results Messages		
	COFFEE	(No column name)
1	Cold Brew	3.333333
2	Espresso	2.000000
3	Turkish Coffee	5.000000

Having

- Having clause is used to filter out grouping records, it's like the WHERE clause. The difference is WHERE clause cannot be used with aggregate functions, whereas Having clause can work with
- Having clause must come after GROUP BY clause and before ORDER BY clause
- Optional

	DRINKER	COFFEE	SCORE
1	Risa	Espresso	2
2	Chris	Cold Brew	1
3	Chris	Turkish Coffee	5
4	Risa	Cold Brew	4
5	Risa	Cold Brew	5

```
SELECT r.COFFEE, AVG(r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY COFFEE
HAVING COUNT(*) >= 3;
```

Results			Messages	
	COFFEE	AVG_RATING		
1	Cold Brew	3.333333		

Order By

- The ORDER BY clause is used to sort the result in ascending or descending order.
- ORDER BY clause sorts the result in ascending order by default. To sort the result by descending order, use the DESC keyword.
- The ORDER BY must come after WHERE, GROUPBY, and HAVING clause if present in the query.

```
SELECT column1, column2,...columnN
FROM table_name
[WHERE]
[GROUP BY]
[HAVING]
ORDER BY column ASC/DESC
```

```
SELECT a.COFFEE
FROM COFFEE_AVG_RATING a
ORDER BY a.AVG_RATING DESC;
```

Offset & Fetch

- The ***OFFSET*** clause specifies the number of rows to skip before starting to return rows from the query.
- The `offset_row_count` can be a constant, variable, or parameter that is greater or equal to zero.
- The ***FETCH*** clause specifies the number of rows to return after the ***OFFSET*** clause has been processed.
- The `offset_row_count` can be a constant or a variable that is greater or equal to one.

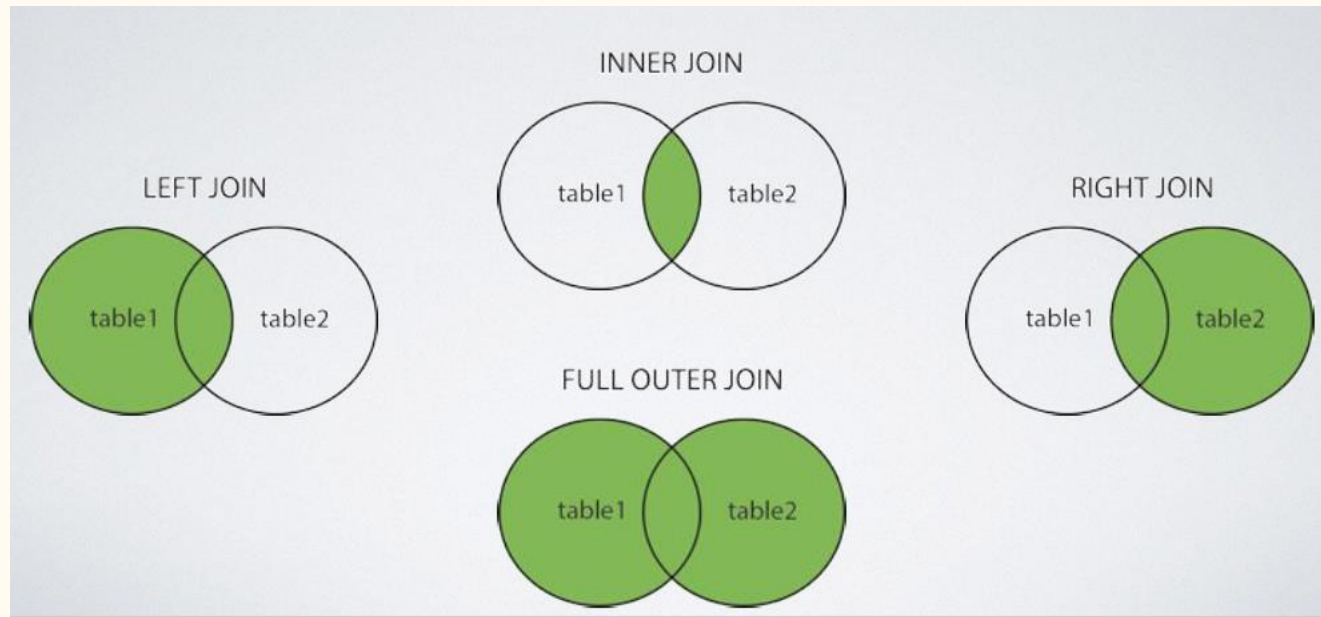
```
SELECT COFFEE
FROM RATES
ORDER BY SCORE DESC
OFFSET 0 ROWS
FETCH NEXT 1 ROWS ONLY
```

	COFFEE
1	Turkish Coffee

Joins

- A JOIN clause is used to combine rows from 2 or more tables, based on a matching column.
- Uses matching data in specified columns to combine or sort data.
- Columns DO NOT have to have the same name.
- Columns DO NOT need to be keys.
- Scope: table to table, table to view, table to synonyms.

Joins



Inner Join

- Inner Join returns records that have matching values in both tables.
- Inner Join selects all rows from both tables as long as there is a match between the columns. If there are records in one table that do not have matches in the other table these records will not be shown.

```
SELECT Orders.OrderID, Customers.CustomerName  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Inner Join

- Query all the students who enroll in the courses and list their enrolled courses

STUDENT

NETID	NAME
rbm2	Risa
abc1	Andre
bcd2	Betty
cde4	Chris

ENROLL

NETID	CRN
abc1	123
abc1	345
cde4	123

COURSE

CRN	NAME
123	COMP 430
234	COMP 533
345	COMP 530

Inner Join

- Query:

```
SELECT *  
FROM STUDENT s  
INNER JOIN ENROLL e ON s.NETID = e.NETID
```

	NETID	NAME	NETID	CRN
1	abc1	Andre	abc1	123
2	abc1	Andre	abc1	345
3	cde4	Chris	cde4	123

Left/Right Join

- The LEFT JOIN keyword returns all records from the left table and the matching records from the right table.
- If there is **no matching record** for the left table, **NULL will be assigned in the result set**.
- It's a good idea to choose one direction (either LEFT or RIGHT) and use it to maintain consistency.

```
SELECT column_name(s)
FROM left_table
LEFT JOIN right_table
ON left_table.column_name = right_table.column_name;
```

Left Join

STUDENT

NETID	NAME
rbm2	Risa
abc1	Andre
bcd2	Betty
cde4	Chris

ENROLL

NETID	CRN
abc1	123
abc1	345
cde4	123

COURSE

CRN	NAME
123	COMP 430
234	COMP 533
345	COMP 530

- Query all students and show the enrollment process

```
SELECT *  
FROM STUDENT s  
LEFT JOIN ENROLL e ON s.NETID = e.NETID
```

```
SELECT *  
FROM ENROLL e  
RIGHT JOIN COURSE c ON e.CRN = c.CRN  
WHERE e.CRN IS NULL
```

	NETID	NAME	NETID	CRN
1	abc1	Andre	abc1	123
2	abc1	Andre	abc1	345
3	bcd2	Betty	NULL	NULL
4	cde4	Chris	cde4	123
5	rbm2	Risa	NULL	NULL

	NETID	CRN	CRN	NAME
1	NULL	NULL	234	COMP 533

Right Join

STUDENT	
NETID	NAME
rbm2	Risa
abc1	Andre
bcd2	Betty
cde4	Chris

ENROLL	
NETID	CRN
abc1	123
abc1	345
cde4	123

COURSE	
CRN	NAME
123	COMP 430
234	COMP 533
345	COMP 530

- Query all the enrollment status in course

```
SELECT *  
FROM ENROLL e  
RIGHT JOIN COURSE c ON e.CRN = c.CRN
```

```
SELECT *  
FROM ENROLL e  
RIGHT JOIN COURSE c ON e.CRN = c.CRN  
WHERE e.CRN IS NULL
```

Results		Messages		
	NETID	CRN	CRN	NAME
1	abc1	123	123	COMP 430
2	cde4	123	123	COMP 430
3	NULL	NULL	234	COMP 533
4	abc1	345	345	COMP 530

Results		Messages		
	NETID	CRN	CRN	NAME
1	NULL	NULL	234	COMP 533

Full Outer Join

- Used to match up tuples from different relations
- Includes all the relations from both sides
- If there is no matching tuple, shows NULL

```
SELECT * FROM t1
LEFT JOIN t2 ON t1.id = t2.id
UNION
SELECT * FROM t1
RIGHT JOIN t2 ON t1.id = t2.id
```

Self Join

- SELF JOIN is used when a JOIN is used on the same table.

FACULTY		
NETID	NAME	MGRNETID
rbm2	Risa	bcd2
abc1	Andre	bcd2
bcd2	Betty	cde4
cde4	Chris	NULL

Self Join

- Query

```
SELECT F.NAME AS EMPLOYEE, MGR.NAME AS MANAGER  
FROM FACULTY F JOIN FACULTY MGR  
ON F.MGRNETID = MGR.NETID
```

	EMPLOYEE	MANAGER
1	Andre	Betty
2	Betty	Chris
3	Risa	Betty

Complete Query Example

```
SELECT DISTINCT column, AGG_FUNC(column_or_expression),  
FROM table1  
JOIN table2  
ON table1.column = table2.column  
WHERE constraint_expression  
GROUP BY column  
HAVING constraint_expression  
ORDER BY column ASC/DESC  
OFFSET count ROWS  
FETCH NEXT num ROWS ONLY
```

Set Operations

- Results are unordered. It could be useful to perform operations on these:

Union

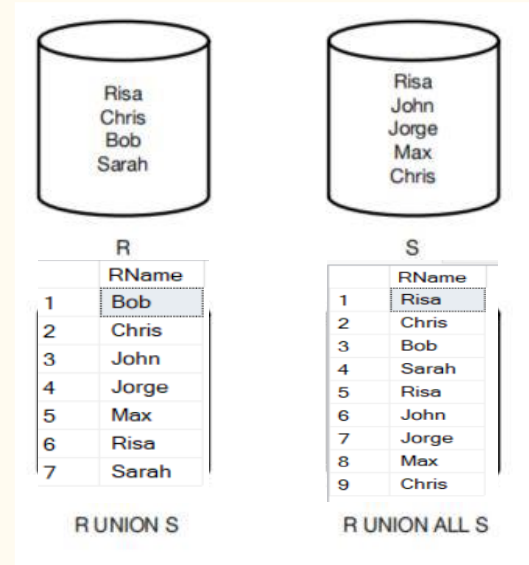
Intersection

Difference

- Different RDBMS provide different levels of support

UNION and UNION ALL

- UNION- eliminates duplicates
- UNION ALL- does NOT eliminate duplicates
- Uses the column names from the first result set
- *Data types* must match
- *Number of attributes* must match

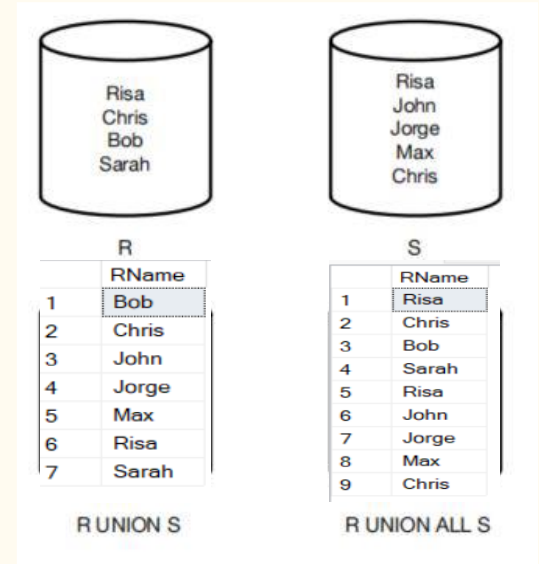


UNION and UNION ALL Example

- R(RName)
- S(SName)

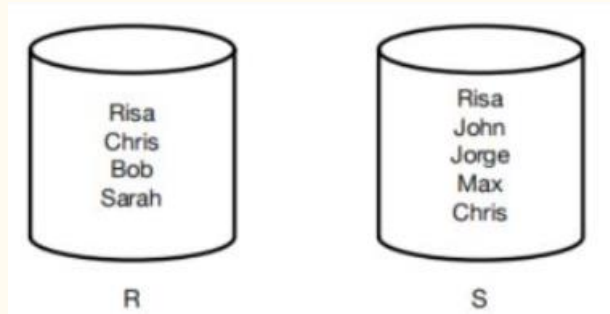
```
-- union
SELECT * FROM R
UNION
SELECT * FROM S
```

```
--union all
SELECT * FROM R
UNION ALL
SELECT * FROM S
```



Intersection

- Intersection Implemented via *INTERSECT*

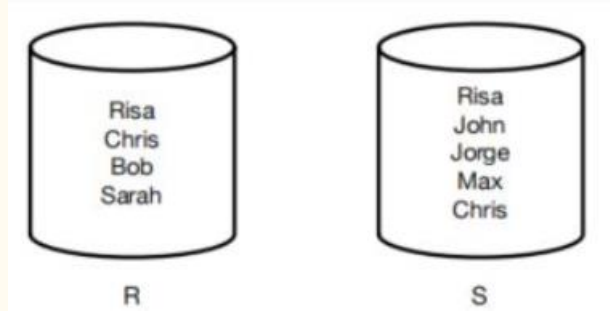


```
SELECT * FROM R  
INTERSECT  
SELECT * FROM S
```

Results		Messages
	RName	
1	Chris	
2	Risa	

Difference

- Difference Implemented via *EXCEPT*
 - Display the values in the first select statement MINUS any values found in the second select statement



```
SELECT * FROM R  
EXCEPT  
SELECT * FROM S
```

Results		Messages
	RName	
1	Bob	
2	Sarah	

Subquery

- A subquery is a query that is nested inside a SELECT, INSERT, UPDATE, DELETE statement or inside another subquery.
- Subqueries must be written in parentheses.
- Subqueries must include the SELECT clause and the FROM clause.

```
SELECT column_name [, column_name ]  
FROM   table1 [, table2 ]  
WHERE  column_name OPERATOR  
      (SELECT column_name [, column_name ]  
       FROM table1 [, table2 ]  
       [WHERE])
```


Where to Put a Subquery

- SELECT clause
 - Can be used to retrieve values in a select clause, but only if they return a single result

```
SELECT Ord.SalesOrderID, Ord.OrderDate,  
       (SELECT MAX(OrdDet.UnitPrice)  
        FROM AdventureWorks.Sales.SalesOrderDetail AS OrdDet  
        WHERE Ord.SalesOrderID = OrdDet.SalesOrderID) AS MaxUnitPrice  
FROM AdventureWorks2008R2.Sales.SalesOrderHeader AS Ord
```

Where to Put a Subquery

- FROM clause
 - Can be used to return an entire table but **must have an alias**
 - Derived Table

```
SELECT X.PRODUCTID
       ,Y.PRODUCTNAME
       ,X.MAX_UNIT_PRICE
FROM (SELECT PRODUCTID
       ,Max(UNITPRICE) AS MAX_UNIT_PRICE
FROM ORDER_DETAILS
GROUP BY PRODUCTID) AS X
INNER JOIN PRODUCTS AS Y
ON X.PRODUCTID = Y.PRODUCTID;
```

Where to Put a Subquery

- WHERE clause
 - Most common use
 - Used to filter results based on another table

```
SELECT productid, productname, unitprice
FROM Production.Products
WHERE unitprice =
(SELECT MIN(unitprice)
FROM Production.Products);
```

View

- A view is often seen as a **virtual table**
- It displays data that you choose, but does not actually hold any data
- Good for security since you can prevent showing extra data
- DML operations just happen on the table.
 - you can modify data on view level, and the source data will be updated as well.

```
CREATE VIEW hiredate_view
AS
SELECT p.FirstName, p.LastName, c.BusinessEntityID, c.HireDate
FROM HumanResources.Employee c
JOIN Person.Person AS p ON c.BusinessEntityID = p.BusinessEntityID ;
GO
```

Variable

- A Transact-SQL local variable is an object that can hold a single data value of a specific type. Variables in *batches* and *scripts* are typically used:
 - As a counter either to count the number of times a loop is performed or to control how many times the loop is performed.
 - To hold a data value to be tested by a control-of-flow statement.
 - To save a data value to be returned by a **stored procedure** return code or function return value.
- User-Defined Variables are displayed with an "@" symbol
- System Variables are displayed with an "@@" symbol

Variable Example

```
DECLARE @MyCounter INT = 1
WHILE (@MyCounter < 10)
BEGIN
    PRINT @MyCounter
    SET @MyCounter = @MyCounter + 1
END
```

```
DECLARE @MyVar VARCHAR(100)

SET @MyVar = 'Bob'

SELECT * FROM R WHERE RName = @MyVar
```

Stored Procedure

- In SQL Server, a stored procedure is a set of T-SQL statements that are compiled and stored in the database. The stored procedure accepts input and output parameters, executes the SQL statements, and returns a result set if any.
- System procedures: System procedures are included with SQL Server and are physically stored in the internal, hidden Resource database and logically appear in the sys schema of all the databases. The system-stored procedures start with the `sp_` prefix.

<https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/system-stored-procedures-transact-sql?view=sql-server-ver16>

User Defined Stored Procedure

- User Defined Stored Procedures are just Stored Procedures, but created by the user
- Contains statements including calling other stored procedures
- Can have different Input and Output Parameters
- Must be recompiled after time or changes

```
-- create or alter procedure
CREATE [OR ALTER] PROCEDURE <pname>
AS
BEGIN
<PROCEDURE BODY/ STATEMENTS>
END

-- call procedure
EXECUTE / EXEC <pname>;
```


User Defined Stored Procedure

- (basic)
- A stored procedure can have zero or more INPUT and OUTPUT parameters.

```
CREATE PROC proc_PeopleName AS
BEGIN
SELECT * FROM R
SELECT * FROM S
END
```

EXEC proc_PeopleName

100 %

Results Messages

	RName
1	Risa
2	Chris
3	Bob
4	Sarah

	SName
1	Risa
2	John
3	Jorge
4	Max
5	Chris

User Defined Stored Procedure

- (SP w/ input parameters)
- Each parameter is assigned a name, and a data type, if no other statement follows, then this parameter is treated as an INPUT parameter

```
CREATE PROC proc_PeopleName_Param(@Name1 VARCHAR(20), @Name2 VARCHAR(20)) AS
BEGIN
SELECT * FROM R WHERE RName = @Name1
SELECT * FROM S WHERE SName = @Name2
END
```

EXEC proc_PeopleName_Param 'Chris', 'Max'

100 %

Results Messages

	RName
1	Chris

	SName
1	Max

User Defined Stored Procedure

- (SP w/ both INPUT and OUTPUT)
- Stored procedures can return a value to the calling program if the parameter is specified as OUTPUT.

```
CREATE PROC proc_GetEmp_Id @empName VARCHAR(10), @Id VARCHAR(10) OUTPUT
AS
BEGIN
    SELECT @Id = NETID
    FROM FACULTY
    WHERE NAME = @empName
END
```

```
DECLARE @ID VARCHAR(10)

EXECUTE proc_GetEmp_Id 'Andre', @Id OUTPUT

PRINT @Id
```

100 %

Messages

abc1

Query Execution

Step 1:

- Parser check query syntax
- Break query to token --> (intermediate files)

Step 2:

- Query Optimizer creates the best possible execution plan based on current resource utilization

Step 3:

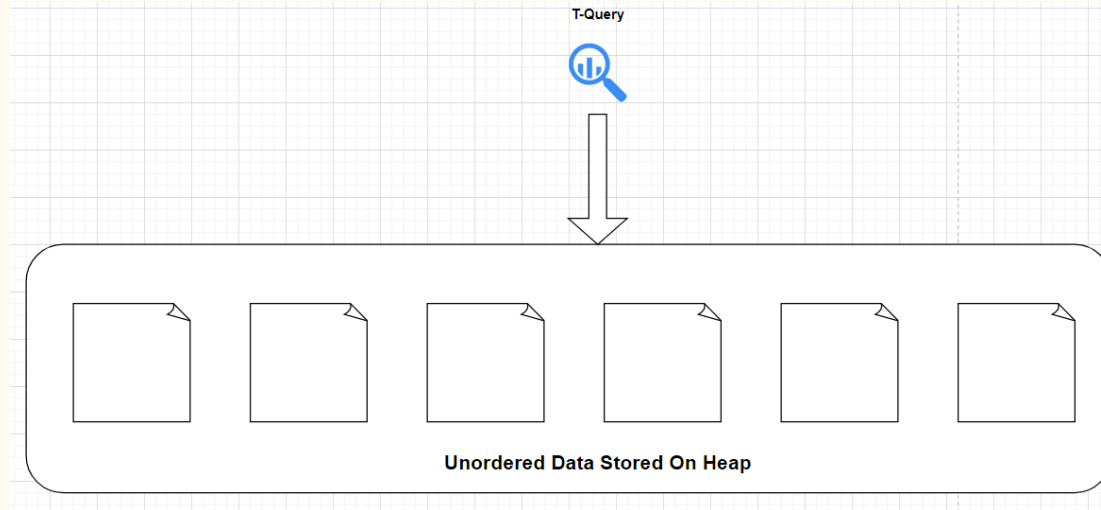
- DB engine --> Run the query

Index

- It's used to sort and **optimize data fetch time**
- Operate similar to index in a book
- When created, an index will create a dynamic Balance Tree (B+ Tree)
- Primary key creates Clustered Index, unique key creates Non-Clustered Index
- Tables without a Clustered Index are called **HEAP** Tables
- Keys \neq Indexes

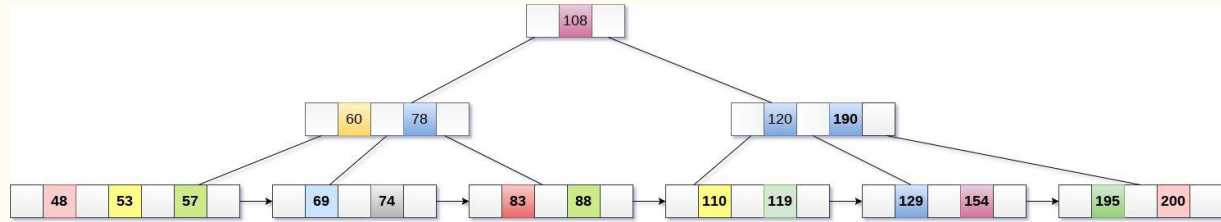
Index

- If a table has no index at all, when new data is inserted, they are added wherever there is free space, and in no particular order.



CLUSTERED INDEX

- Composed of 3 main levels
 - Root Level
 - Intermediate Level
 - Leaf Page Level
- Each Node is about 8KB in size
 - 8060B for data
 - 132B for pointers
 - 8192B in Total



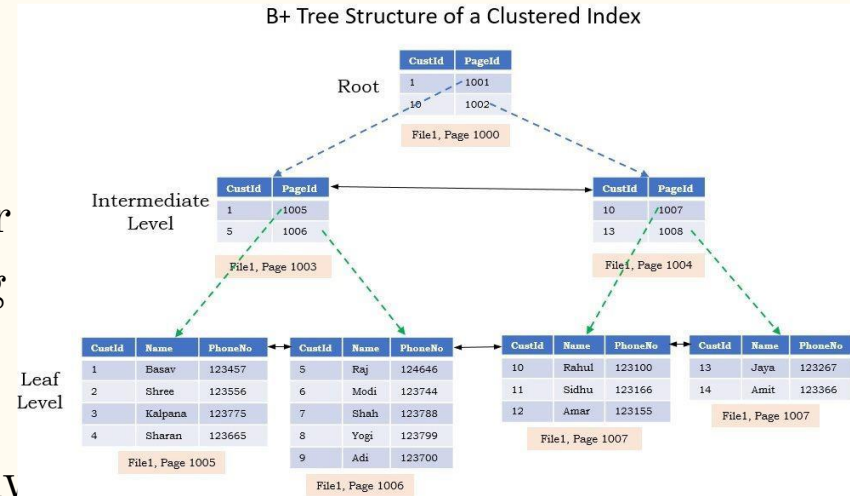
Each Index created will have a Balance tree structure to be used, but the type of Index will determine how data is stored in a Balance Tree

Clustered Indexes will store data in Leaf Pages and sort them based on the Key values of the column you choose.

Non-Clustered Indexes will **NOT** store data in the Leaf Pages, instead they'll point to the rows they're referencing

CLUSTERED INDEX

- A clustered index will physically move the data from the table into its Balance Tree
- The data is now matching physically and logically
- Data is sorted based on ascending order for the column chosen, this becomes the clustering key
- This is why there can only be 1 Clustered
- Index on a table, data can only be physically sorted and stored once



NON-CLUSTERED INDEX

- Since Non-Clustered Indexes do not physically move or store data, there can be many on a single table.
 - Currently up to 999 different Indexes
- A Non-Clustered Index on a table with a Clustered Index must now grab data from the B-Tree of the CI.
- So data will come up through the Root of the CI and fall into the Leaf Pages of the NCI

Leaf node of a nonclustered index on LastName

Adams	3
Douglas	4
Jones	1
Smith	2

Leaf node of a clustered index on EmployeeID

1	Jones	John
2	Smith	Mary
3	Adams	Mark
4	Douglas	Susan

Questions

- How does Index improve the performance?
Find Index vs. Table Scan
- Will Index always improve the performance?
Maintain the index

Questions?